Katana4D
Handbuch

©Neuronics AG, 2001-2008. All rights reserved
Version: 5.0.3.0

Neuronics AG, Technoparkstrasse 1, CH-8005 Zürich
Tel: +41 44 445 16 40, Fax: +41 44 445 16 44
www.neuronics.ch
info@neuronics.ch

Inhaltsverzeichnis

Einleitung....5
Weitere Dokumente....5
Installation....5
Darstellungen in der Dokumentation....5

1 Softwarebeschreibung....6

1.1 Überblick....7
1.1.1 Start / Anmeldung....7
1.1.2 Einrichtungsassistent....7
1.1.3 Benutzerverwaltung....8
1.1.3.1 Benutzer....8
1.1.3.2 Gruppen....8
1.1.4 Einstellungen....10
1.1.4.1 Sprache ändern....10
1.1.4.2 Verbindung herstellen/trennen....10
1.1.4.3 Datendatei....10
1.1.5 Struktur....11
1.1.5.1 Projekt / Programm....11
1.1.5.2 Anwendung....11
1.1.6 Handsteuerung....12
1.1.6.1 Motoren-Navigation....12
1.1.6.2 Raum-Navigation....12
1.1.6.3 Tool-Navigation....13
1.1.6.4 Weitere Befehle....13
1.1.6.5 Hauptjustierung....13

1.2 Katana-Einrichtung....14
1.2.1 Katana Allgemein....14
1.2.2 Greifer....15
1.2.2.1 Sensorgreifer....15
1.2.2.2 Andere Greifer....15
1.2.3 Sensorik....16
1.2.4 Parameter....17
1.2.4.1 Katana-Parameter....17
1.2.4.2 Controller-Parameter....18
1.2.5 Firmware....18

1.3 Anwendung....19
1.3.1 Aufbau der Anwendung....19
1.3.2 Anwendung erstellen....20
1.3.3 Hilfsfunktionen....21

1.4 Projekt....22
1.4.1 Allgemein....23
1.4.2 Raster....24
1.4.2.1 Allgemeine Definitionen....24
1.4.2.2 Erweitertes Endraster....26
1.4.2.3 Rasterpunkte löschen....26
1.4.2.4 Rasterpunkte teachen und Rasterkoordinaten berechnen....28
1.4.3 Punkte....30
1.4.4 Wege....32
1.4.5 Trajektorie....36
1.4.6 Benutzervariablen....39
1.4.7 Projekt-Funktionen....40

1.5 Programm....42
1.5.1 Allgemein....43
1.5.2 Aufbau des Programms....44
1.5.3 Abbruch des Programms....45
1.5.4 Katana4D Script....46
1.5.4.1 Kommentare in Katana4D....46
1.5.4.2 Katana-Befehle....46
1.5.4.3 Variablen....46
1.5.5 Startscript und Endscript....48
1.5.6 Sequenzen....49
1.5.7 Benutzermethoden....51
1.5.7.1 Lokale Benutzermethoden....51
1.5.7.2 Globale Benutzermethoden....51
1.5.7.3 Ereignisabhängige Benutzermethoden....53
1.5.8 Optionen....55
1.5.9 Suche....58
1.5.10 Programm-Funktionen....59
1.5.11 Stand-alone....61
1.6 Schnittstellen....64
1.6.1 ModbusTCP....64
1.6.2 National Instruments Digital I/O Card....65
1.6.3 Datei Kommunikation....68
1.6.4 TCP/IP....69
1.6.5 Siemens S7 – Plugin....70
1.6.6 Web-Services....72

2 Befehlsbeschreibung....73

2.1 Aktionsbefehle....79
2.1.1 Allgemeine Aktionsbefehle....80
2.1.2 Anfahren von Positionen im Raum....84
2.1.3 Anfahren von Punkten im Raster....89
2.1.4 Anfahren von Punkten im Raum....91
2.1.5 Abfahren der Wege....95
2.1.6 Abfahren der Trajektorien....96
2.1.7 Lesen und Speichern....97
Punkte....97
Raster....99
Benutzervariablen....102
2.2 Zustandsabfragen....105
2.3 Schnittstellen....110
2.3.1 National Instruments IO Card....110
2.3.2 Datei-Kommunikation....112
2.3.3 Serielle Kommunikation....114
2.3.4 TCP/IP....116
2.3.5 ModBusTCP....119
2.3.6 Siemens S7 - Plugin....123
2.3.7 Controlbox....134
2.4 Programmablauf-Steuerung....136
2.4.1 Input/Output....139
2.4.2 Program....141
2.5 Mathematische Funktionen....144
2.6 Datum & Zeit....149

2.7 String.....152
2.8 Array.....157
2.9 Operatoren.....163

Einleitung

Diese Anleitung informiert Sie über die Inbetriebnahme Ihres Katana Roboters mit Hilfe der Katana4D Steuerungssoftware.

Wir empfehlen Ihnen, diese Anleitung sorgfältig zu lesen. Sie werden feststellen, dass ein Katana einfach programmiert und in Betrieb genommen werden kann.

Weitere Dokumente

Diese Anleitung wird durch die folgenden Dokumentationen vervollständigt:

1. Katana4D Quickstart Guide
2. Katana 450 Benutzerhandbuch
3. Stand-alone Quickstart Guide

Installation

Um die Software zu installieren, legen Sie die CD in ihr CD-ROM-/DVD-ROM-Laufwerk ein und warten, bis das Autostart-Menü erscheint. Klicken Sie auf Installieren und folgen Sie den Anweisungen auf dem Bildschirm.

Hinweis

Falls das Autostart-Menü nicht automatisch erscheint, können Sie den Arbeitsplatz öffnen und anschliessend das CD-ROM-/DVD-ROM-Laufwerk, in dem die CD eingelegt ist, doppelklicken. Führen Sie die Datei 'setup.exe' mit einem Doppelklick aus. Die Installation startet.

Darstellungen in der Dokumentation

In diesem Handbuch werden folgende Elemente immer gleich dargestellt:

- *Eingabefelder* und *Checkboxen*
- Schaltflächen
- ***Register*** und ***Fenster***
- Normaler Text
- 'Text für die Eingabe im Formular'

1 Softwarebeschreibung

1.1 Überblick

1.1.1 Start / Anmeldung

Nach dem Start der Software öffnet sich folgender Bildschirm:

Um sich anzumelden, wird das Menü 'Katana' geöffnet und der Menüeintrag 'Login' ausgewählt oder die Tastenkombination 'Strg+L' gedrückt. Im erscheinenden Fenster muss der Benutzername und das Passwort eingegeben werden. Beim ersten Start der Software stehen die Benutzer 'User' und 'Administrator' zur Verfügung. Der Administrator verfügt über alle Rechte, der User hingegen hat nur eingeschränkten Zugriff (siehe Abschnitt 'Benutzerverwaltung').

1.1.2 Einrichtungsassistent

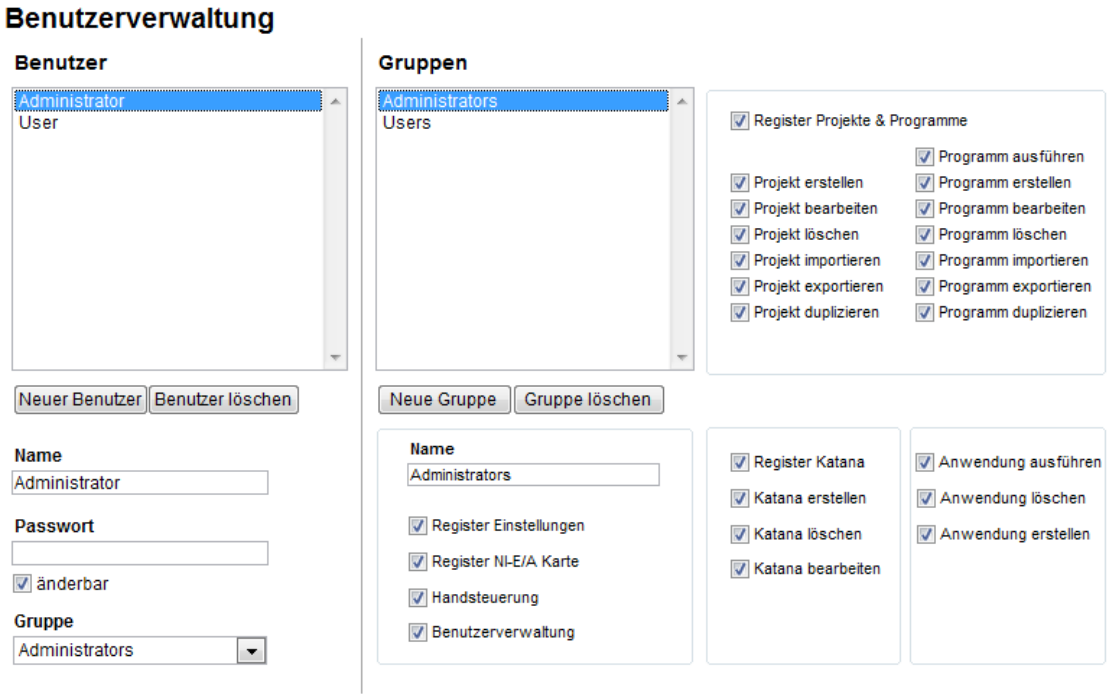
Nach dem ersten Start von Katana4D öffnet sich der Einrichtungsassistent. Der Assistent hilft beim Einrichten der Software.

Auf der ersten Seite muss die Katana-Konfiguration ausgewählt werden. Bei Achse 6 ist auszuwählen, ob auf dem Roboter ein Greifer oder ein Dreher verwendet wird. Im Feld *Name* kann dem Katana ein Name gegeben werden.

Auf der letzten Seite kann das Administrator-Passwort gesetzt werden. Im zweiten Feld muss das Passwort bestätigt werden.

1.1.3 Benutzerverwaltung

Katana4D bietet dem Administrator umfangreiche Möglichkeiten zur Benutzerverwaltung. Diese befindet sich im Register ***Benutzerverwaltung***.



Standardmässig werden die Benutzerkonten 'User' und 'Administrator' sowie die Gruppen 'Users' und 'Administrators' erstellt.

1.1.3.1 Benutzer

Ein neuer Benutzer kann durch Klicken auf die Schaltfläche Neuer Benutzer erstellt werden.

- Der Name kann im Feld *Name* geändert werden.
- Das Passwort kann im Feld *Passwort* festgelegt werden. Zudem kann eingestellt werden, ob das Passwort für den Benutzer änderbar sein soll. Wenn diese Eigenschaft ausgewählt ist, kann der Benutzer über das Menü 'Extras' -> 'Passwort ändern' ein neues Passwort setzen.
- Unter *Gruppe* kann der Benutzer einer Gruppe zugeordnet werden. Die Benutzerrechte werden entsprechend den Rechten der Gruppe gesetzt.

Die beiden Benutzer 'User' und 'Administrator' können nicht gelöscht werden. Andere können durch Betätigen der Schaltfläche Benutzer löschen entfernt werden.

1.1.3.2 Gruppen

Benutzer werden einer Gruppe zugeordnet. Sie besitzen die Rechte dieser Gruppe. Folgende Rechte können eingestellt werden:

Recht	Beschreibung
Register NI-E/A Karte	Das Register der NI-Karte ist verfügbar (wenn eine NI-

Recht	Beschreibung
	Karte angeschlossen ist)
Handsteuerung	Die Handsteuerung kann verwendet werden
Benutzerverwaltung	Der Benutzer darf Benutzer und Gruppen verwalten, Rechte bestimmen wie auch Passwörter ändern
Register Projekte & Programme	Das Register Projekte & Programme ist verfügbar
Projekt erstellen	Projekte dürfen erstellt werden
Projekt bearbeiten	Projekte dürfen bearbeitet werden
Projekt löschen	Projekte dürfen gelöscht werden
Projekt importieren	Projekte dürfen importiert werden
Projekt exportieren	Projekte dürfen exportiert werden
Projekt duplizieren	Projekte dürfen dupliziert werden
Programm ausführen	Programme dürfen ausgeführt werden (aus dem Register 'Projekte & Programme')
Programm erstellen	Programme dürfen erstellt werden
Programm bearbeiten	Programme dürfen bearbeitet werden
Programm löschen	Programme dürfen gelöscht werden
Programm importieren	Programme dürfen importiert werden
Programm exportieren	Programme dürfen exportiert werden
Programm duplizieren	Programme dürfen dupliziert werden
Register Katana	Das Register Katana ist verfügbar
Katana erstellen	Es dürfen weitere Katanas erstellt werden
Katana löschen	Katanas dürfen gelöscht werden
Katana bearbeiten	Katanas dürfen bearbeitet werden
Anwendung ausführen	Anwendungen dürfen ausgeführt werden
Anwendung löschen	Anwendungen dürfen gelöscht werden
Anwendung erstellen	Anwendungen dürfen erstellt werden

Hinweis

Vorsicht: Wenn der Gruppe 'Administrators' das Recht zur Benutzerverwaltung entzogen wird, kann dies nicht mehr rückgängig gemacht werden.

Änderungen in der Benutzerverwaltung werden erst bei der nächsten Anmeldung angewendet.

1.1.4 Einstellungen

1.1.4.1 Sprache ändern

Die Sprache kann vor dem Anmelden geändert werden. Durch Auswahl des Menüeintrags 'Extras' -> 'Sprache ändern' erscheint ein Dialog, in dem die entsprechende Sprache ausgewählt werden kann. Damit die Sprache korrekt geladen wird, ist ein Neustart von Katana4D notwendig.

1.1.4.2 Verbindung herstellen/trennen

Die Verbindung zur Controlbox kann über den Menüeintrag 'Kommunikation' -> 'Verbindung herstellen' hergestellt werden. Wenn die Verbindung steht, verfärbt sich die Statusanzeige rechts unten im Hauptfenster grün und zeigt die Meldung 'Online' an.

Alternativ kann die Verbindung auch durch einen Klick auf die Statusanzeige hergestellt oder getrennt werden.

Auch wenn Katana4D nicht mit der Controlbox verbunden ist, stehen dennoch die meisten Funktionen zur Verfügung. Das heisst, es kann auch ohne bestehende Verbindung programmiert werden.

1.1.4.3 Datendatei

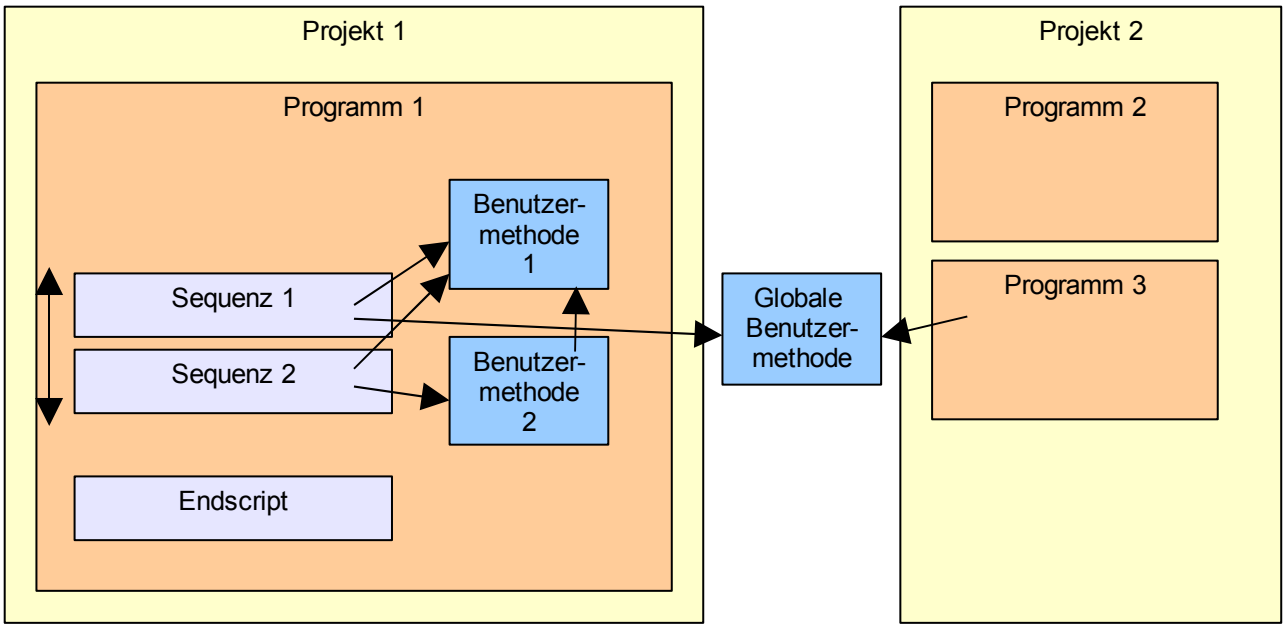
Katana4D legt alle Daten (Katana-Einstellungen, Projekte, Programme etc.), die seit dem ersten Start von Katana4D erstellt wurden, in einer Datendatei mit der Endung “4DD“ ab. Die Standard-Datendatei “Katana4D.4DD“ ist im Installationsverzeichnis von Katana4D im Ordner “Database“ abgelegt. Soll eine Sicherung gemacht werden, muss nur diese Datei gespeichert werden. Um einzelne Projekte oder Programme auszutauschen, steht im Register ***Projekte & Programme*** eine Import/Export-Funktion zur Verfügung.

Soll eine andere Datendatei geladen oder der Pfad der aktuellen Datendatei geändert werden, kann dies über den Menüeintrag 'Extras' -> 'Datenbank' erfolgen. Es öffnet sich ein Standard-Dialog, in dem die 4DD-Datei ausgewählt werden kann.

Mit Neue Datendatei erstellen wird eine neue Datenbank erstellt und geladen. Die alte Datei wird nicht überschrieben. Sie kann später wieder über den gleichen Menüpunkt geladen werden. Wird eine, noch leere Datendatei geladen, erscheint der Einrichtungsassistent wie bei der Installation von Katana4D.

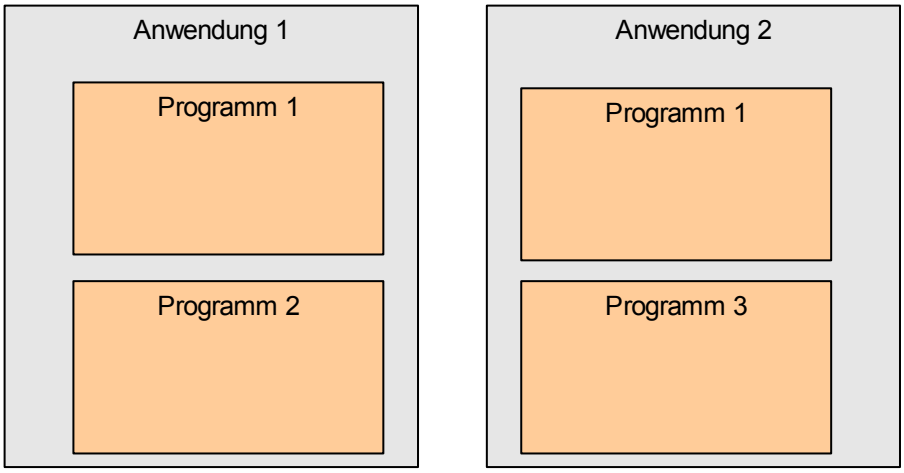
1.1.5 Struktur

1.1.5.1 Projekt / Programm



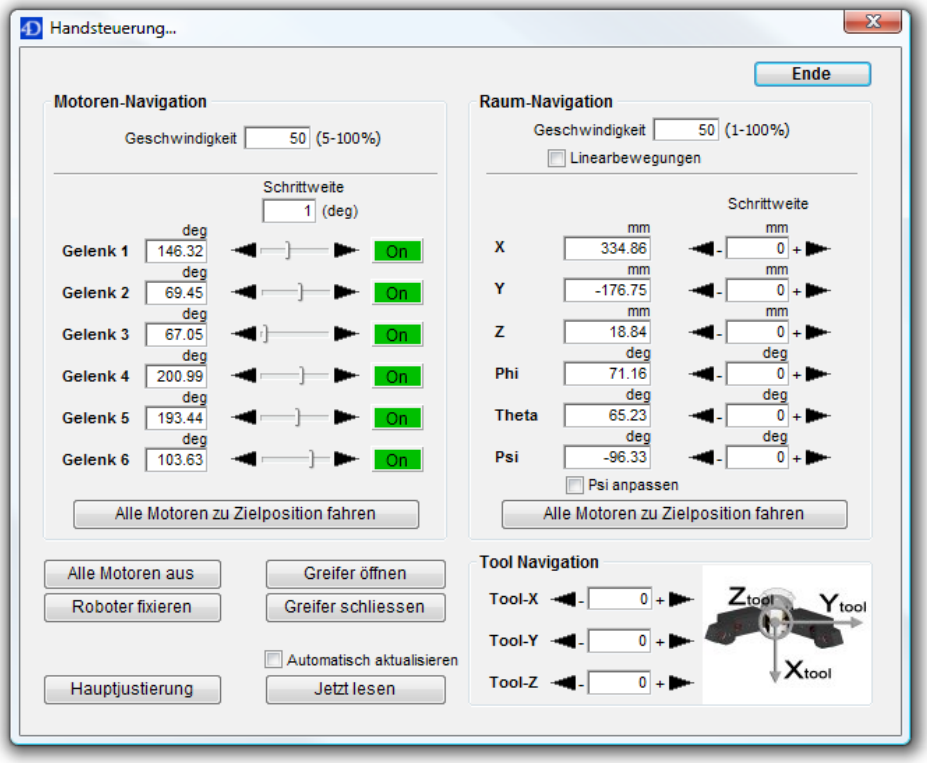
Ein Projekt beinhaltet Definitionen, die in einem Programm verwendet werden können. Ein Programm wiederum ist ein Bestandteil eines Projekts. Das Programm greift auf den Katana zu, der dem Projekt zugeordnet ist, sowie wie auch auf die gespeicherten Werte (Raster oder Punkte) des Projekts.

1.1.5.2 Anwendung



Die Anwendung ist eine unabhängige Ebene, zu der Programme hinzugefügt werden können. Eine Anwendung wird immer als ganze Gruppe ausgeführt.

1.1.6 Handsteuerung



1.1.6.1 Motoren-Navigation

Durch Klicken auf einen Pfeil bewegt sich die entsprechende Achse des Katana um die eingestellte Schrittweite (in Grad) vor- oder rückwärts. Die Zielpositionen der einzelnen Achsen können auch in die entsprechenden Felder geschrieben werden. Durch Klicken der Schaltfläche Alle Motoren zu Zielposition fahren, wird der Roboter in Bewegung gesetzt. Dabei sollte darauf geachtet werden, dass der Katana nicht in ein Hindernis fährt.

Tipps

- Als Schrittweite Werte zwischen '1' und '10' Grad wählen.
- Sicherstellen, dass keine Hindernisse (oder der Boden) im Weg sind, so dass der Katana sicher an den gewünschten Ort fahren kann.

1.1.6.2 Raum-Navigation

Der Katana ist auch in einem dreidimensionalen, kartesischen Koordinatensystem steuerbar. Dabei interessiert in erster Linie die Bewegung des Tool Center Points (TCP), nicht die Bewegung der einzelnen Achsen. Der TCP ist ein gedachter Referenzpunkt, der sich an geeigneter Stelle am Werkzeug befindet, also zum Beispiel vorne am Greifer. Der Nullpunkt des Basiskoordinaten-systems befindet sich standardmässig in der Mitte der zweiten Achse. Die Z-Achse ist die Senkrechte. Zur eindeutigen Steuerung des Roboters bedarf es der Eulerwinkel 'Phi', 'Theta' und 'Psi'. Sie geben die Ausrichtung des Greifers zum angefahrenen Raumpunkt an. Die Einheit für 'X', 'Y' und 'Z' ist Millimeter, diejenige für 'Phi', 'Theta' und 'Psi' ist Grad.

In den Feldern auf der rechten Seite (*Schrittweite*) kann die Änderung eingegeben werden. Über die Schaltfläche Alle Motoren zu Zielposition fahren fahren alle Achsen gleichzeitig an die angegebene Position.

Mit der Auswahl der Eigenschaft *Linearbewegung* wird die Zielposition linear angefahren, das heisst der TCP bewegt sich entlang einer Geraden im Raum.

1.1.6.3 Tool-Navigation

Im Bereich Tool-Navigation kann der Roboter im Tool-Koordinatensystem bewegt werden. Tool-Z ist in Richtung des Tools und Y entlang der Finger.

Die Geschwindigkeit sowie die Auswahl der Linearbewegung werden aus der Raum-Navigation übernommen.

1.1.6.4 Weitere Befehle

Alle Motoren aus	Um den Roboter von Hand zu bewegen, können Sie die Motoren per Knopfdruck ausschalten. **VORSICHT: Roboter vor dem Ausschalten der Motoren festhalten.**
Roboter fixieren	Zur erneuten Fixierung des Roboters.
Greifer öffnen	Öffnet den Greifer (wenn Motor6 ein Greifer ist).
Greifer schliessen	Schliesst den Greifer (wenn Motor6 ein Greifer ist).
Hauptjustierung	Führt eine Hauptjustierung durch. Siehe Abschnitt Hauptjustierung.
Jetzt lesen	Aktualisiert die Motor- und Raumpositionen. Wenn das Auswahlfeld 'Automatisch aktualisieren' ausgewählt ist, werden die Werte laufend vom Roboter übertragen.

1.1.6.5 Hauptjustierung

Nach jedem Stromunterbruch des Roboters ist eine Kalibrierung vorzunehmen. Durch betätigen der Schaltfläche Hauptjustierung wird diese ausgeführt. Alle Achsen werden mit langsamer Geschwindigkeit an die mechanischen Anschläge gefahren und auf den Vorgabewert referenziert.

Diese Funktion sollte nach einem Stromausfall, nachdem der Roboter in ein Hindernis gefahren ist oder wenn Positionen nicht mehr genau angefahren werden können, angewendet werden.

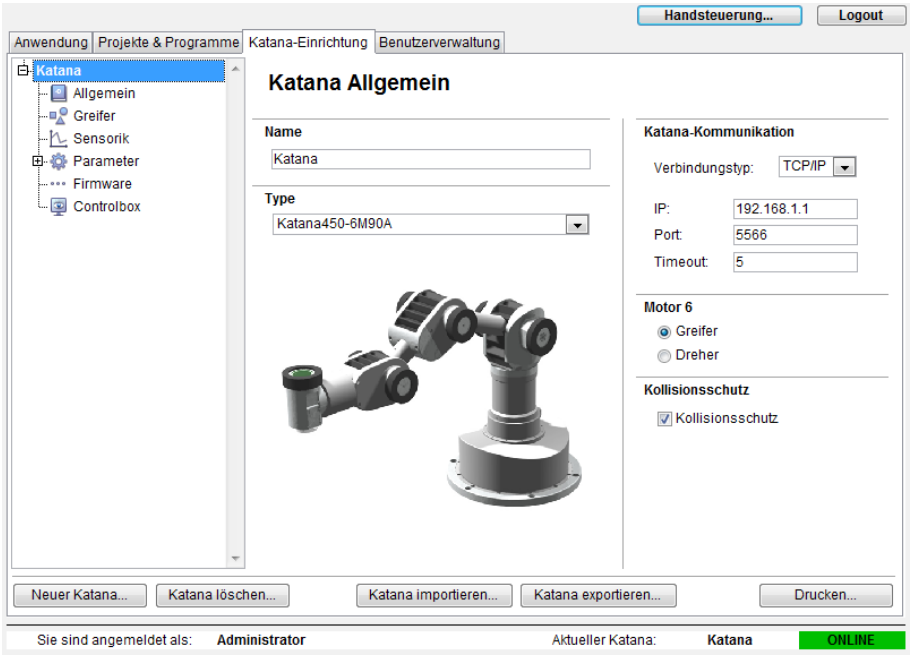
1.2 Katana-Einrichtung

Im Register ***Katana-Einrichtung*** werden alle Einstellungen vorgenommen, die den Roboterarm betreffen. In der Liste auf der linken Seite werden die bestehenden Katana-Profile aufgelistet. Für jedem Katana kann es verschiedene Einstellungen geben, die in den folgenden Abschnitten erläutert werden.

Mit der Schaltfläche Neuer Katana... kann ein neues Katana-Profil erstellt werden. Auch wenn nur ein Katana zur Verfügung steht, kann es nützlich sein, zwei Profile mit verschiedenen Parametern anzulegen. Sollte eine andere Parametrierung nötig sein, kann ein anderes Katana-Profil geladen werden.

Durch Drücken der Schaltfläche Drucken... kann das ausgewählte Katana-Profil ausgedruckt werden. Über Katana importieren... und Katana exportieren... kann das Profil abgespeichert, bzw. geladen werden.

1.2.1 Katana Allgemein



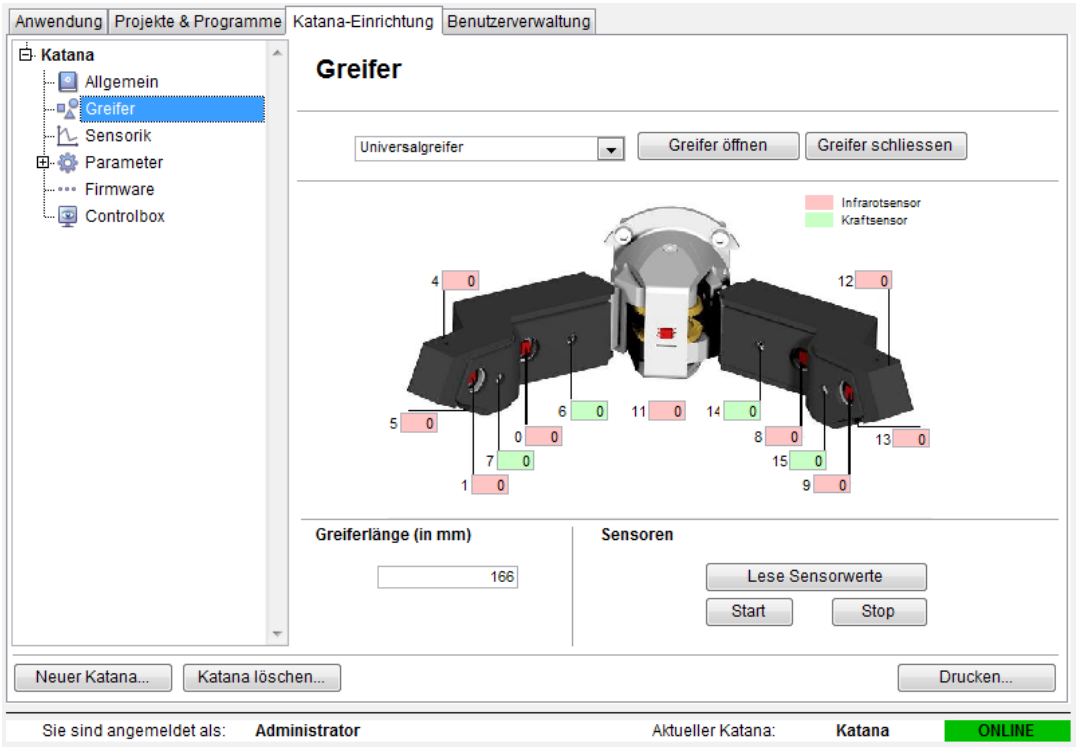
Im Feld *Name* wird der Name des Katana-Profils festgelegt. Im Auswahlfeld darunter muss der Typ, der verwendet wird, ausgewählt werden. Die richtige Auswahl ist sehr wichtig, denn aus einer falschen Typenwahl können unvorhersehbare Bewegungen resultieren.

Wird der Roboter über ein Netzwerkkabel angeschlossen, muss im Feld *Verbindungstyp* 'TCP/IP' ausgewählt werden. Darunter muss die IP-Adresse des Roboters eingegeben werden. Bei einer Verbindung über die USB-Schnittstelle wird 'USB' ausgewählt. Falls die USB-Verbindungskonfiguration des Roboters geändert wurde, muss 'TCP/IP' ausgewählt werden und die IP-Adresse der USB-Verbindung eingetragen werden.

Kollisionsschutz legt fest, ob der Roboter bei einer Kollision stehen bleiben soll. Aus Sicherheitsgründen wird empfohlen, diese Eigenschaft immer aktiviert zu haben.

Motor 6 legt fest, ob der sechste Motor als Greifer oder Dreher verwendet wird. Wenn dieser Motor nicht vorhanden ist, muss die Eigenschaft 'Greifer' ausgewählt werden.

1.2.2 Greifer



1.2.2.1 Sensorgreifer

Der Sensorgreifer ist pro Finger mit vier Infrarot- und zwei Kraftsensoren ausgerüstet. Zu-sätzlich steht ein weiterer Infrarotsensor auf dem Greifergrundkörper zur Verfügung.

Mit Lese Greifer-Sensorwerte werden die Werte der einzelnen Sensoren einmal gelesen und in die entsprechenden Felder eingetragen. Durch drücken auf Start werden die Sensorwerte solange gelesen und angezeigt bis Stop gedrückt wird.

Im Feld *Greiferlänge (in mm)* wird die Länge des installierten Greifers angegeben.

Mit Greifer öffnen und Greifer schliessen können Sensorwerte, die von einem Gegenstand geliefert werden, ermittelt werden.

1.2.2.2 Andere Greifer

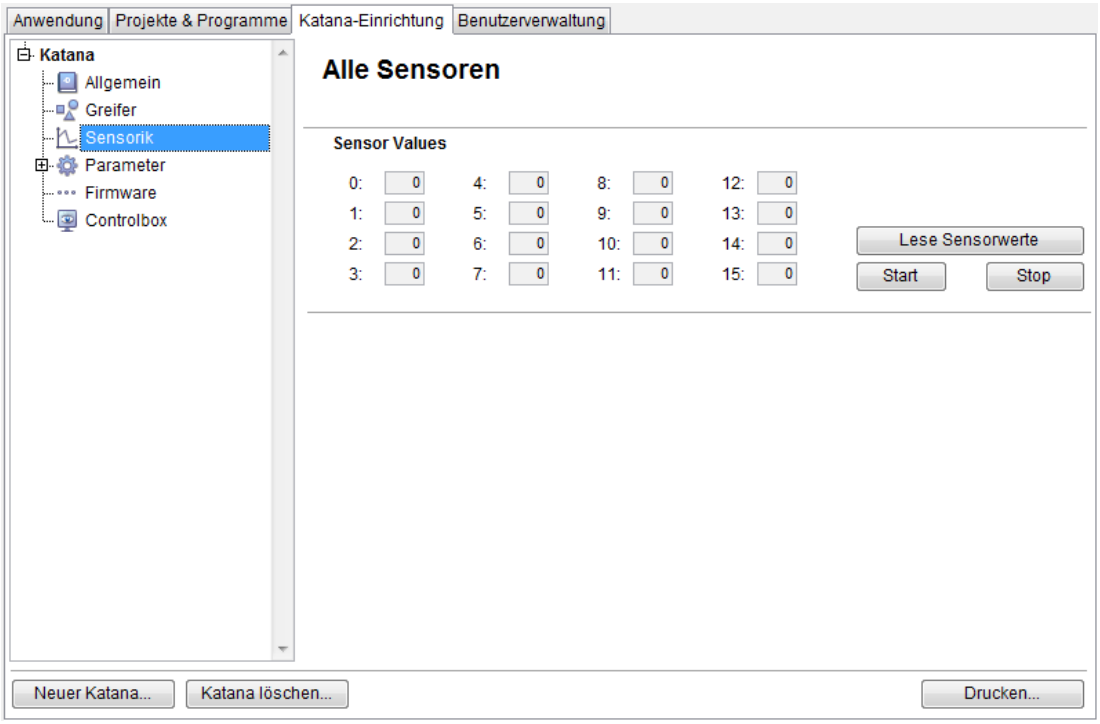
Pendelgreifer

Dieser Greifertyp verfügt über einen weiteren Freiheitsgrad. Die Gravitation wird genutzt, um die Greifbacken in vertikaler Lage zu stabilisieren. Dadurch können Objekte, unabhängig vom Armwinkel, immer gleich gegriffen werden.

Vakuumgreifer

Der Vakuumgreifer ist ein Greifer, der pneumatisch betrieben wird. Dieser Greifer benötigt keinen Motor als Antrieb.

1.2.3 Sensorik

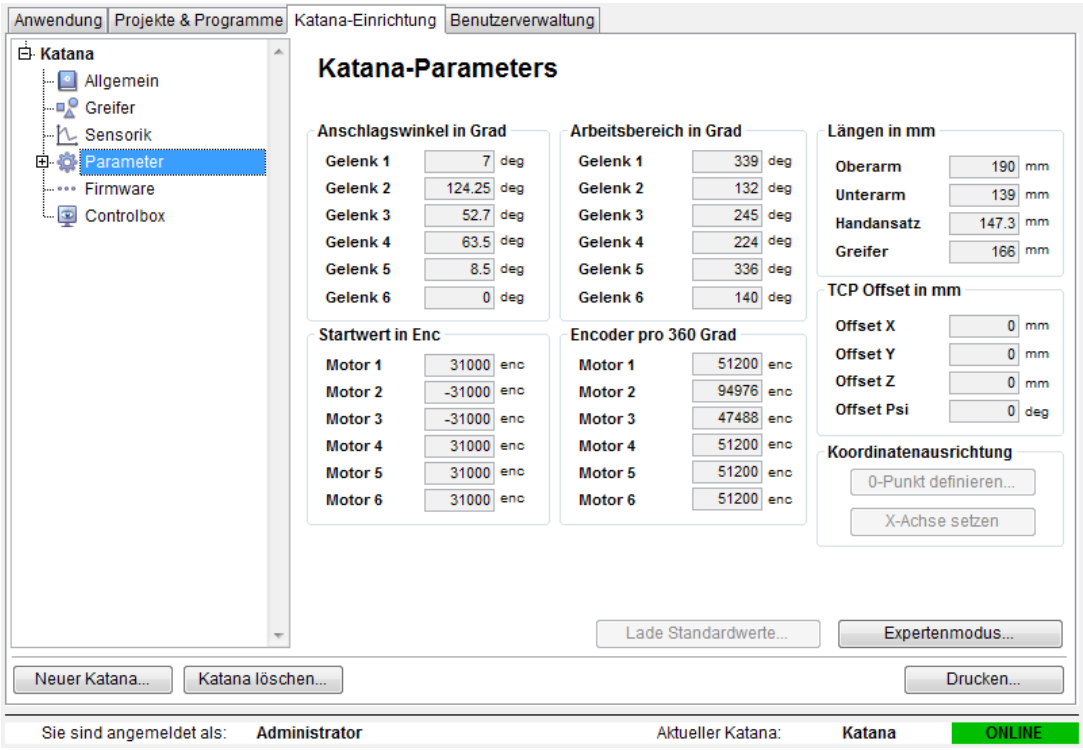


In diesem Fenster können alle Sensorsignale ausgelesen werden.

Mit Lese Sensorwerte werden die einzelnen Sensoren einmal gelesen. Die gemessenen Werte werden in die entsprechenden Felder geschrieben. Durch drücken auf Start werden die Sensorwerte kontinuierlich gelesen und angezeigt bis Stop gedrückt wird.

1.2.4 Parameter

1.2.4.1 Katana-Parameter



Im Register ***Katana-Parameters*** können die Parameter des Roboters festgelegt werden. Um die Parameter zu ändern, muss in den Expertenmodus gewechselt werden. Vorsicht: Parameter sollten nur geändert werden, wenn dies unbedingt erforderlich ist. Eine Beschreibung der Parameter ist im 'Katana Handbuch' zu finden.

Mit Lade Standardwerte... werden die Ausgangswerte geladen.

X-Achse setzen

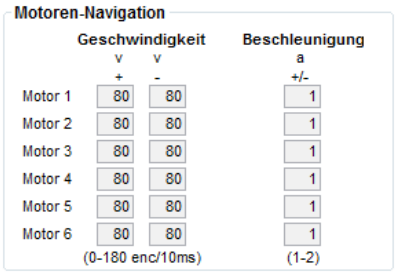
Mit X-Achse setzen wird das Koordinatensystem ausgerichtet. Die positive X-Achse wird auf die aktuelle Position des Roboters gesetzt. Der Anschlagswinkel vom Gelenk 1 wird automatisch umgerechnet.

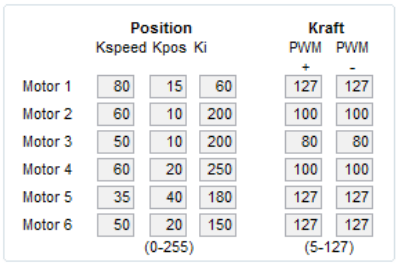
0-Punkt definieren

Der 0-Punkt des Koordinatensystems kann ebenfalls verschoben werden. Dazu wird der Roboter in eine beliebige Position im Raum gefahren und 0-Punkt definieren... gedrückt. Im geöffneten Fenster kann die aktuelle Position mit Werte übernehmen als 0-Punkt geteacht werden. Nach der Übernahme der Werte erscheint die Position in den drei Delta-Feldern. Mit Zurücksetzen wird der zuvor gespeicherte Wert wieder übernommen. Auf 0 zurücksetzen... setzt alle Felder auf 0, der fixe 0-Punkt wird nicht verschoben.

1.2.4.2 Controller-Parameter

Controller-Parameter sind Werte, die beim Roboter auf jedem Achscontroller gespeichert sind. Diese Parameter sind bereits optimiert und sollten nur verändert werden, wenn dies unbedingt erforderlich ist. Dazu muss zuerst in den Expertenmodus... gewechselt werden.





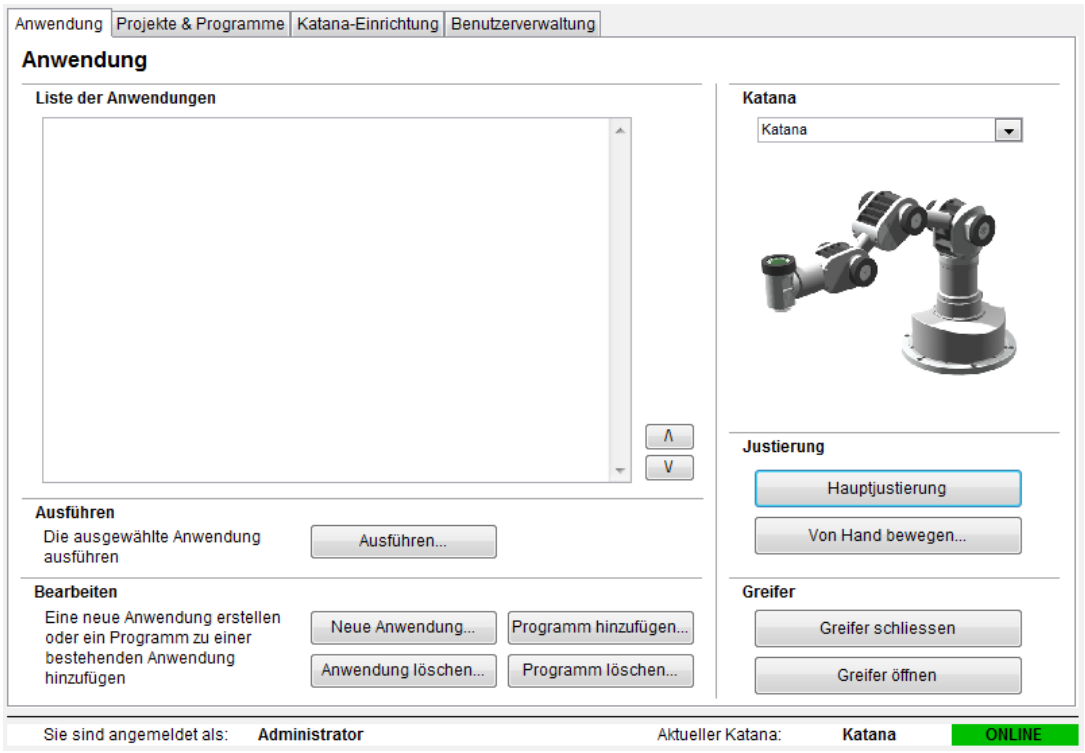
Eine Beschreibung der Parameter ist im 'Katana450 Manual' zu finden

1.2.5 Firmware

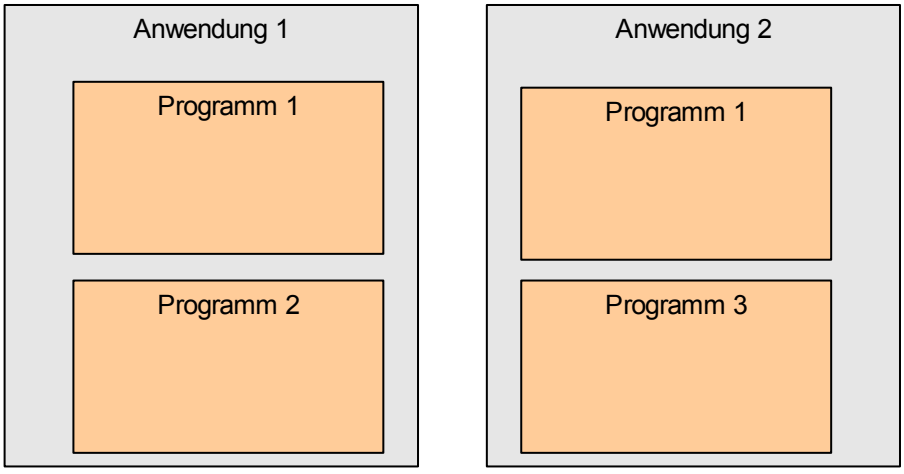
Die aktuell geladenen Firmware-Versionen des Roboters werden hier angezeigt. Über die Schaltfläche Firmware lesen können die Versionen neu ausgelesen werden.

1.3 Anwendung

Die Anwendung ist eine unabhängige Ebene, zu der Programme hinzugefügt werden können.



1.3.1 Aufbau der Anwendung

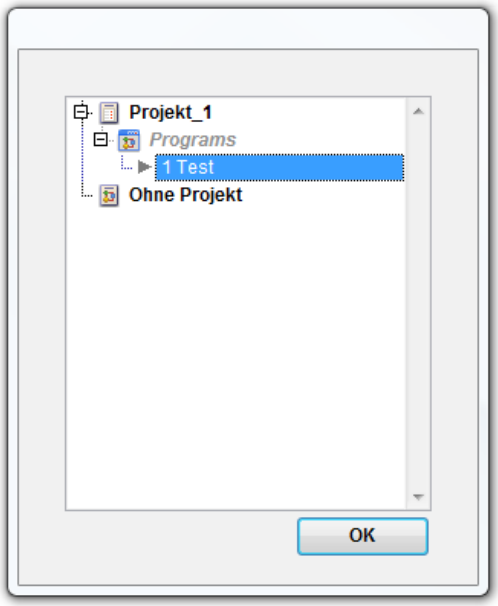


Eine Anwendung besteht aus verschiedenen Programmen. Es können beliebig viele Programme, mit Ausnahme von mehreren Programmen des gleichen Projekts, hinzugefügt werden. Mehrere Programme aus dem gleichen Projekt können nicht hinzugefügt werden, weil Programme, die auf den gleichen Katana zugreifen, nicht parallel arbeiten können.

1.3.2 Anwendung erstellen

Durch das Betätigen der Schaltfläche Neue Anwendung... wird eine neue Anwendung erstellt. Nachdem ein Name angegeben wurde, erscheint die Anwendung in der Liste.

Wenn der Anwendung ein Programm hinzugefügt werden soll, muss diese ausgewählt und die Schaltfläche Programm hinzufügen... gedrückt werden.



Im erscheinenden Fenster werden alle Projekte mit den dazugehörigen Programmen aufgelistet.

Um ein Programm zu entfernen, muss dieses in der Liste markiert und über die Schaltfläche Programm löschen... gelöscht werden.

Mithilfe der Schaltflächen ∧ und ∨ können Programme einer Anwendung verschoben werden. Die Ausführungsreihenfolge entspricht der Reihenfolge, in der sie in der Liste erscheinen. Das bedeutet, dass ein Programm, das Initialisierungen durchführt, als erstes gestartet werden muss. Aus diesem Grund muss es in der Liste zuoberst stehen.

Programme können auch mittels Drag & Drop verschoben werden.

Mit Anwendung löschen... kann eine komplette Anwendung entfernt werden.

Eine Anwendung wird immer als Ganzes ausgeführt. Nach dem Drücken der Schaltfläche Ausführen werden die Programme der Reihe nach gestartet. Nachdem alle Programme gestartet worden sind, verschwindet die Schaltfläche Ausführen... und die Schaltfläche Stop erscheint. Durch betätigen dieser Schaltfläche wird die Anwendung gestoppt und alle Programme beendet. Zwei Anwendungen gleichzeitig auszuführen, ist nicht möglich.

1.3.3 Hilfsfunktionen

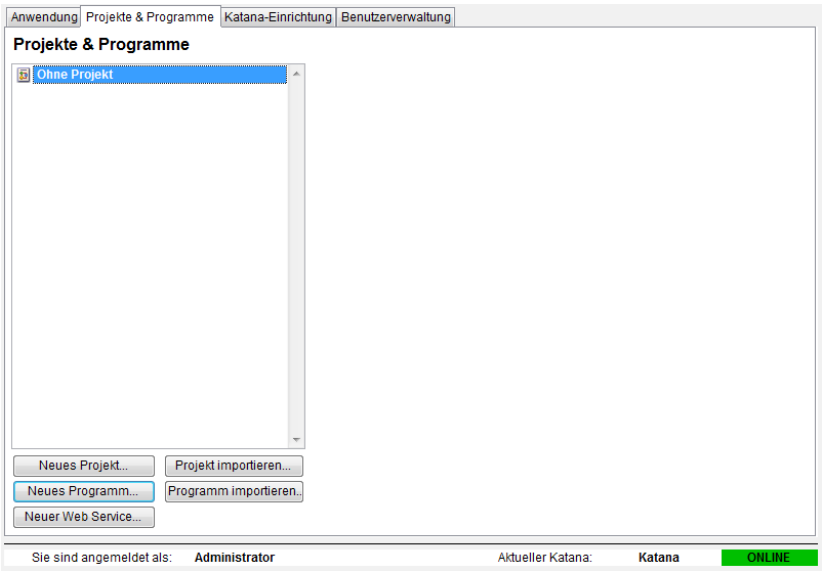
Unter *Katana* wird der aktuell ausgewählte Roboter aufgeführt. Falls mehrere Katanas existieren, kann der entsprechende Roboter über die Auswahlliste gewählt werden.

Im unteren Teil stehen folgende Funktionen zur Verfügung:

Hauptjustierung	Führt eine Hauptjustierung des ausgewählten Roboters durch.
Von Hand bewegen...	Der Roboter schaltet alle Motoren aus, damit er von Hand bewegt werden kann. Am Schluss werden wieder alle Motoren fixiert, so dass der Roboter wieder stabil ist.
Greifer öffnen	Öffnet den Greifer (nur verfügbar wenn Motor6 ein Greifer ist).
Greifer schliessen	Schliesst den Greifer (nur verfügbar wenn Motor6 ein Greifer ist).

1.4 Projekt

Ein Projekt beinhaltet Definitionen, die in einem Programm verwendet werden können.



Ein neues Projekt wird durch das Klicken der Schaltfläche Neues Projekt... erstellt.

Ein Name für des Projektes muss angegeben werden. Dieser kann später nicht mehr geändert werden.

Es gibt zwei Möglichkeiten, ein Projekt zu erstellen bzw. zu bearbeiten:

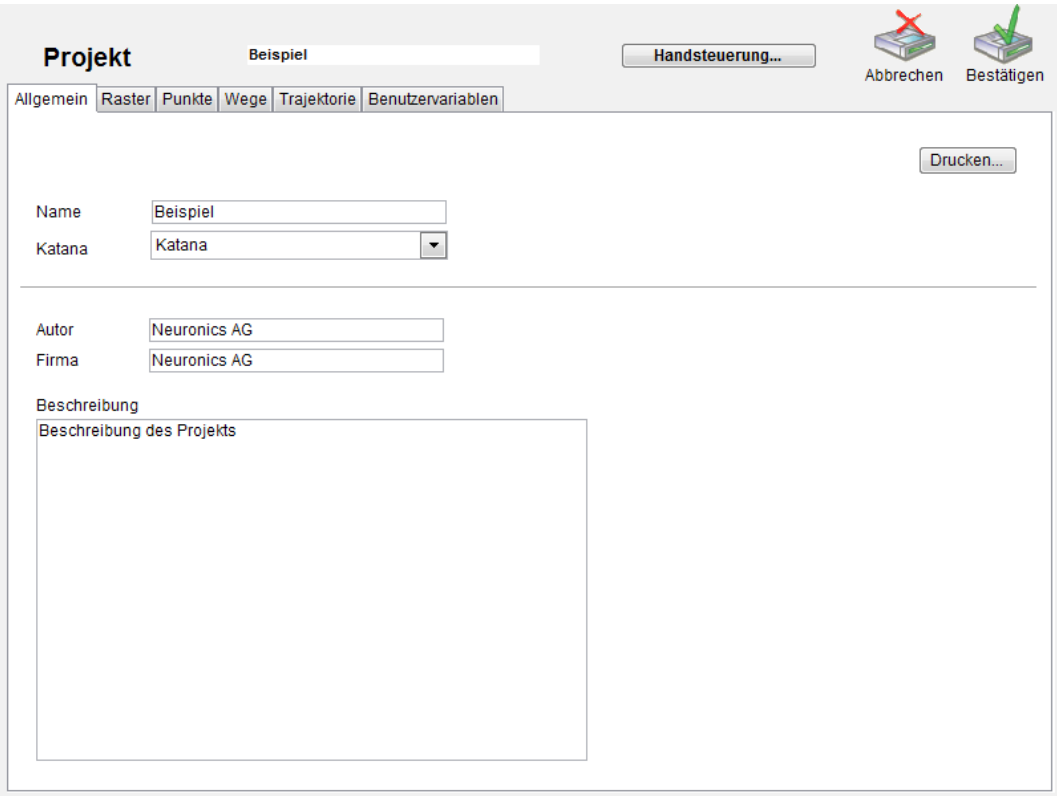
- mit Hilfe vom Assistenten, *Assistent verwenden* auswählen (siehe Kap. Katana4D Assistent)
- manuell

Der Assistent ist für den Einstieg oder für eine Präsentation geeignet, da man sehr schnell einen Ablauf erstellen kann. Für eine tiefere Programmierung müssen jedoch einzelne Bereiche des Projektes betrachtet und manuell geändert werden.

Das Projekt besteht aus folgenden Registern:

- Allgemein
- Raster
- Punkte
- Wege
- Trajektorie
- Benutzervariablen

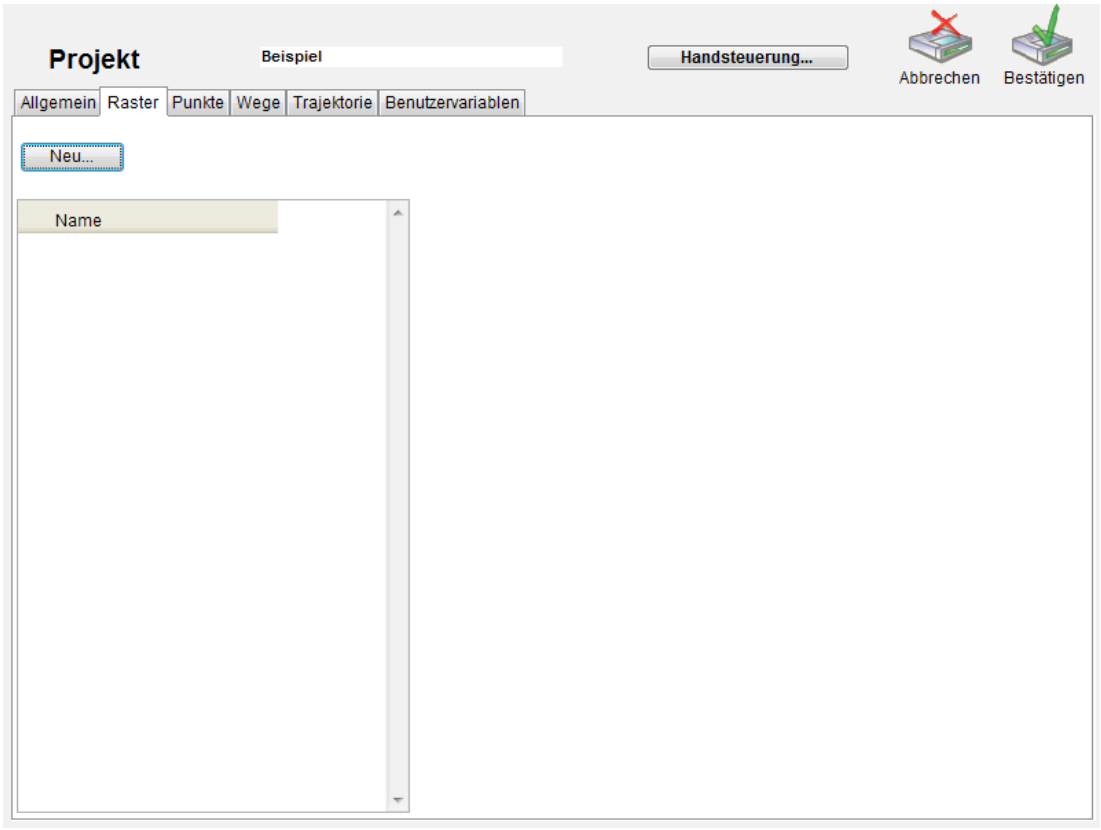
1.4.1 Allgemein



Die Felder *Beschreibung*, *Autor* und *Firma* können verwendet werden, um das Projekt zu beschreiben. Obwohl diese Felder nicht obligatorisch sind, ist es empfehlenswert sie auszufüllen. Im Feld *Katana* kann der Roboter ausgewählt werden, der für dieses Projekt verwendet soll.

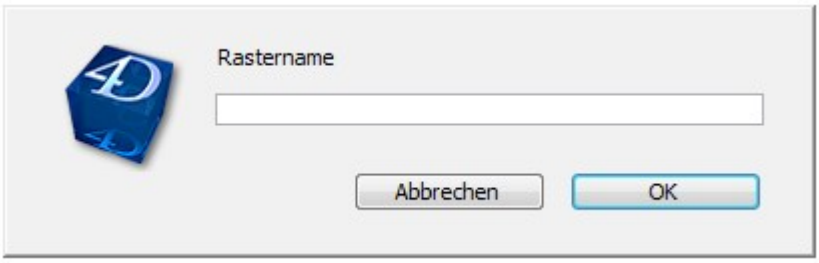
1.4.2 Raster

1.4.2.1 Allgemeine Definitionen



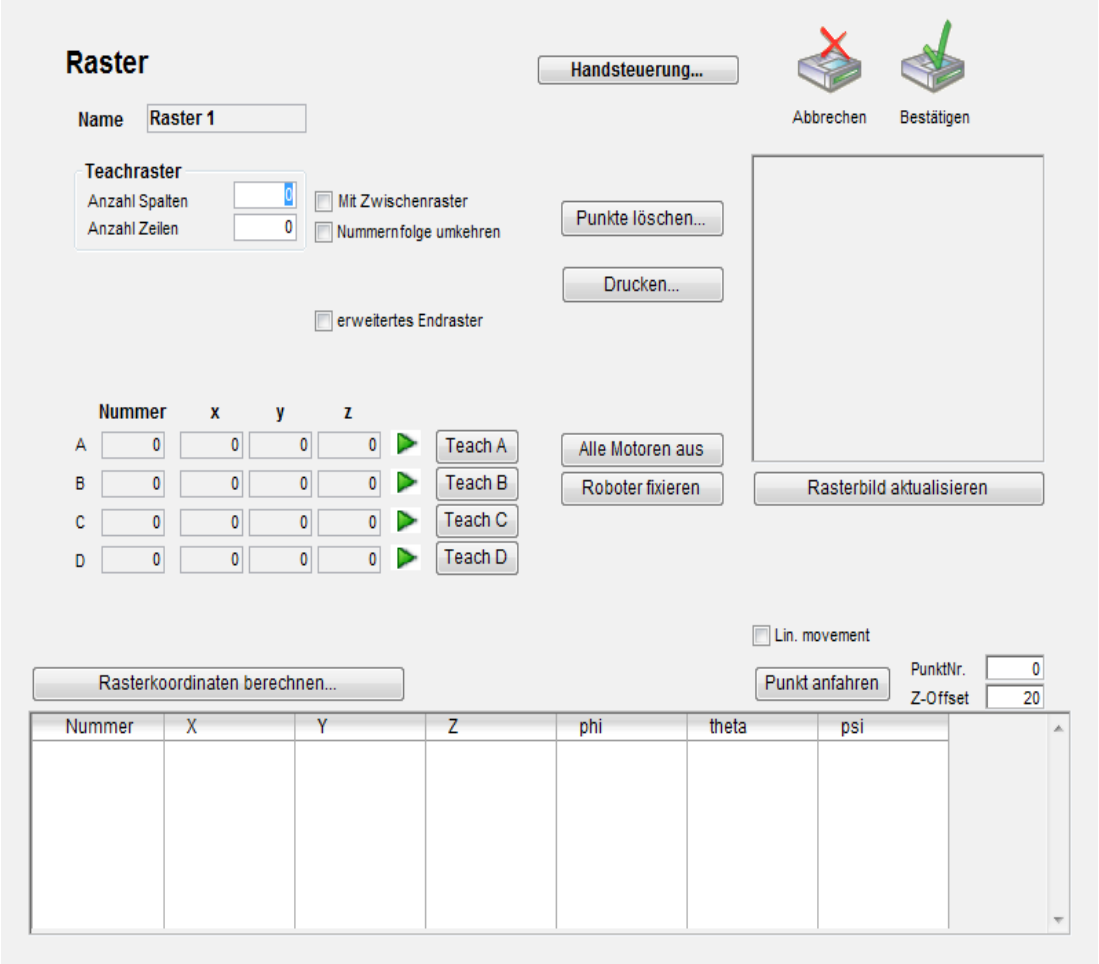
In einem Projekt können Punkte definiert werden, die frei im Raum liegen. Sind sie aber so angeordnet, dass die Abstände zwischen den Punkten gleich sind und alle Punkte auf einer Ebene liegen, kann die Rasterfunktion verwendet werden.

Ein Projekt kann beliebig viele Raster enthalten. Mit Neu... wird ein Raster erstellt.



Der Name des Rasters wird im Feld *Rastername* zugewiesen. Wird ein Name gewählt, der bereits für ein anderes Raster verwendet wird, erscheint eine Fehlermeldung. Der Name kann später nicht mehr geändert werden.

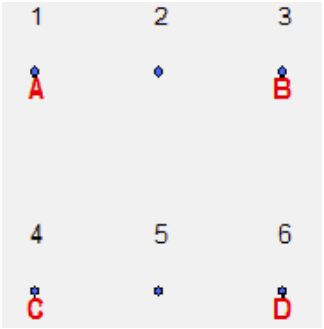
Falls eine Änderung des Namens dennoch nötig sein sollte, kann das Raster kopiert und der gewünschte Name zugewiesen werden.



Das Raster wird auf Grund von vier Aussenpunkten A, B, C und D berechnet. Die Dimension, die Grösse des Rasters, wird in den Feldern *Anzahl Spalten* und *Anzahl Zeilen* eingetragen. Die Zwischenpunkte werden dann von 1 bis [Anzahl Spalten * Anzahl Zeilen] nummeriert. Mit der Schaltfläche Rasterbild aktualisieren werden die Punkte des Raster graphisch angezeigt. Das Bild oberhalb der Schaltfläche dient dem Benutzer sich im Rasterfeld orientieren zu können. Es beeinflusst die Funktionalität nicht.

Mit der Checkbox *Nummernfolge umkehren* kann die Nummerierung der Zwischenpunkte umgekehrt werden.

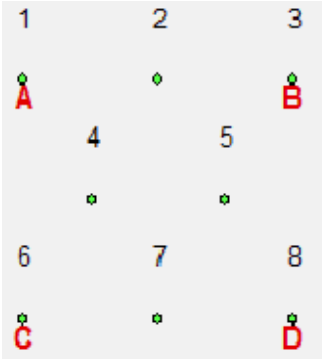
Wird ein Haken in der Checkbox *Mit Zwischenraster* gesetzt, werden zwischen den einzelnen Punkten Zwischenpunkte erstellt. Dementsprechend wird die Anzahl der Punkte und deren Nummerierung erhöht.



Das Bild des Rasters wird nicht automatisch aktualisiert. Dies muss manuell mit Rasterbild aktualisieren bewirkt werden.

1.4.2.2 Erweitertes Endraster

Falls das wirkliche Aufnahmeraster grösser sein soll als das Raster, das sich aus den Teach-Punkten ergeben würde, oder wenn die Eckpunkte nicht geteacht werden können, weil es in den Ecken keine Punkte gibt, dann wird das erweiterte Endraster gebraucht.

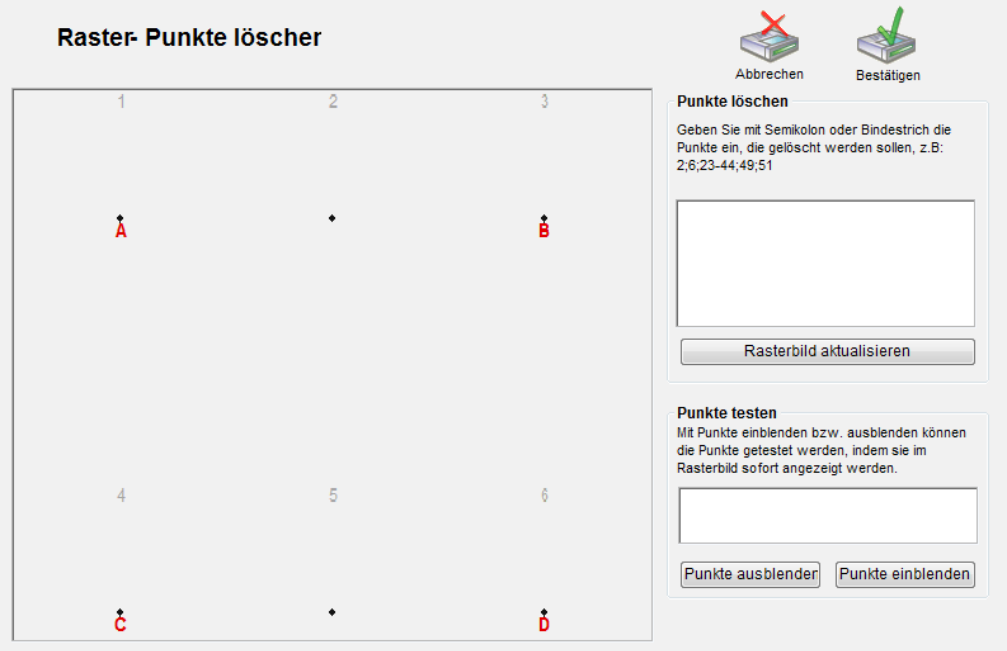


Durch Auswählen der Checkbox erweitertes Endraster erscheinen die Felder *Anzahl Spalten* und *Anzahl Zeilen*, die für die Angabe der Dimension des erweiterten Rasters gebraucht werden.

Die Eckpunkte des “Teachrasters“ werden geteacht, die des Endrasters werden berechnet.

1.4.2.3 Rasterpunkte löschen

Falls nicht alle Rasterpunkte angefahren werden können oder sollen, können die entsprechenden Punkte gelöscht werden. Mit Punkte löschen... erscheint das Fenster **Rasterpunkte löschen**.

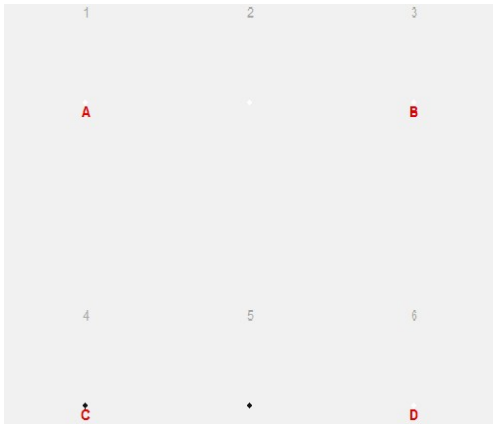


Auf dem Bild links erscheinen die Punkte des Rasters in schwarzer Farbe. Im Feld *Punkte löschen* können einzelne Punkte, mit Semikolon getrennt, oder mehrere, durch Bindestrich verbunden, gelöscht werden. Bevor die Punkte gelöscht werden, können sie im Feld *Punkte testen* mit Punkte ausblenden und Punkte einblenden getestet werden. Damit wird sichergestellt, dass wirklich nur die gewünschten Punkte gelöscht werden.

Beispiel

Im Feld *Punkte löschen* wird “1-3; 6“ eingegeben.

Die Punkte 1, 2, 3 und 6 werden gelöscht und weiss dargestellt.



Die Rasterpunkte werden aber nicht sofort aus der Liste gelöscht, sondern erst bei der Neuberechnung des Rasters.

Mit Bestätigen wird das Fenster geschlossen und die Änderung gespeichert.

1.4.2.4 Rasterpunkte teachen und Rasterkoordinaten berechnen

Bis jetzt wurde das Raster definiert, aber die Punkte noch nicht konkret erstellt. Bevor die Zwischenpunkte berechnet werden können, müssen die Eckpunkte “geteacht“ werden. Dazu wird der Roboter zum Punkt A gefahren und mit Teach A der erste Eckpunkt gespeichert. Vor dem Teachen kann der Punkt mit Handsteuerung... noch justiert werden. Im Hintergrund wird auch der Anfahrtswinkel des Punktes gespeichert. Auf gleiche Weise werden die restlichen drei Eckpunkte B, C und D geteacht.

Nach dem Teachen der Eckpunkte folgt die Berechnung der Zwischenpunkte. Mit Raster-koordinaten berechnen... werden die Zwischenpunkte des Rasters erstellt und in der Liste angezeigt.

Nummer	X	Y	Z	phi	theta	psi
1	346.93	-94.04	-26.05	88.49	89.34	-90.31
2	345.38	-1.32	-21.72	86.4	89.08	-89.71
3	343.82	91.41	-17.39	84.31	88.81	-89.11
4	363.9	102.7	-121.92	83.5	87.66	-88.18
5	348.1	11.27	-138.28	85.69	89.02	-89.02

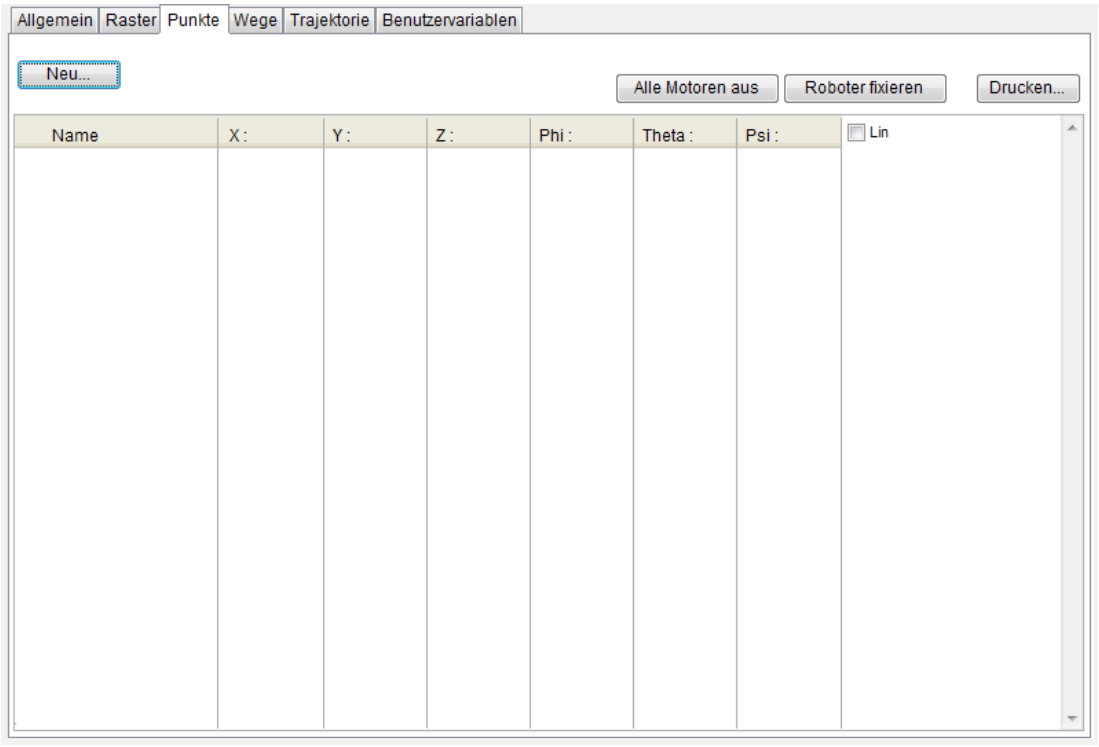
TIPP: Bei den Eckpunkten sollte darauf geachtet werden, dass sich die Achsen nicht zu nahe am jeweiligen mechanischen Anschlag befinden. Sonst kann es sein, dass die Rasterpunkte nicht berechnet werden können, da einzelne Positionen der Achsen ausserhalb des Arbeitsbereiches liegen. Dies gilt insbesondere für Achse 6, falls kein Greifer ausgewählt wurde.

Die einzelnen Punkte können zum Testen mit einem Z-Offset (optional; Standard 20mm) mit Punkt anfahren angefahren werden. Sollte der angefahrene Punkt nicht genau sein, kann er korrigiert werden. Dazu wird der Punkt in der Liste ausgewählt und mit der Handsteuerung... verbessert. Anschliessend kann die korrigierte Position durch das Klicken von gespeichert werden.

Die Definition des Rasters und dessen Punkte können über Drucken... ausgedruckt werden.

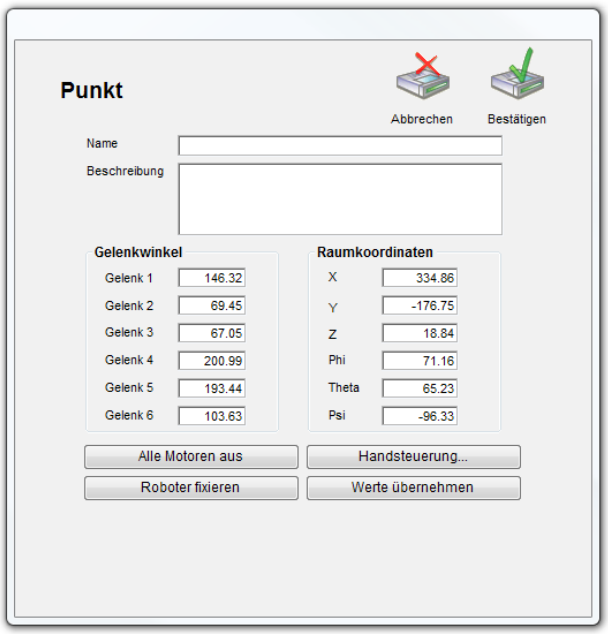
Das Raster ist nun erstellt. Durch das Drücken der Schaltfläche Bestätigen schliesst sich das Fenster und das neu erstellte Raster erscheint in der Rasterliste.

1.4.3 Punkte



Falls die anzufahrenden Punkte beliebig im Raum liegen, werden sie im Register ***Punkte*** definiert.

In der Liste können beliebig viele Punkte definiert werden. Innerhalb eines Projekts müssen die Namen der Punkte eindeutig sein.



Mit Neu... erscheint ein Fenster zum Einfügen von Punkten.

Das Feld *Name* ist obligatorisch. Der Roboter wird von Hand zum gewünschten Punkt gebracht und mit Werte übernehmen wird die aktuelle Position gespeichert.

Mit Handsteuerung... kann der Roboter genauer positioniert werden. Durch das Drücken von Bestätigen schliesst sich das Fenster und der neu eingefügte Punkt erscheint in der Liste.

	Name	X :	Y :	Z :	Phi :	Theta :	Psi :	Lin
>>	Startposition	351.56	232.42	-14.93	-168.42	80.09	174.56	

Bei einem Klick in die Zeile erscheinen folgende Hilfsfunktionen:

- Öffnen des Eingabefensters ***Punkt*** für die Bearbeitung des Punktes.
- Anfahren des Punktes (Linearbewegung falls Lin angekreuzt ist).
- Speichern der aktuellen Position (überschreibt die alten Werte).
- Punkt löschen (Vorsicht!).

Die einzelnen Punkte in der Liste können, wie oben beschrieben, für Testzwecke angefahren werden. Der ganze Ablauf, d.h. die Reihenfolge, wie die verschiedenen Punkte angefahren werden sollen und was dort gemacht werden soll, wird im Programm definiert.

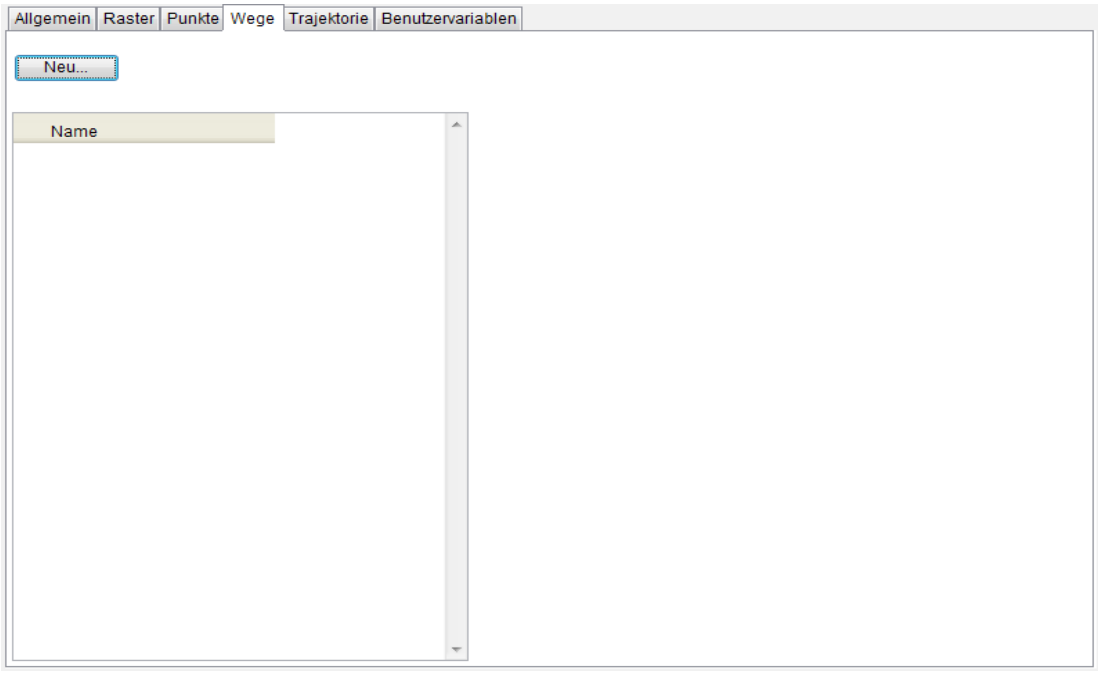
Die Punkte der Liste können über die Schaltfläche Drucken ausgedruckt werden.

Punkteliste... Page 1

Projekt Beispiel

Name	X	Y	Z	Phi	Theta	Psi
Endposition	332.3	-80.13	-154.64	87.87	90.38	-89.85
Startposition	351.56	232.42	-14.93	-168.42	80.09	174.56

1.4.4 Wege



Mit der Weg-Funktion kann der Katana von Hand geführt und seine Bewegung aufgezeichnet werden. Anschliessend kann der Weg abgefahren werden.

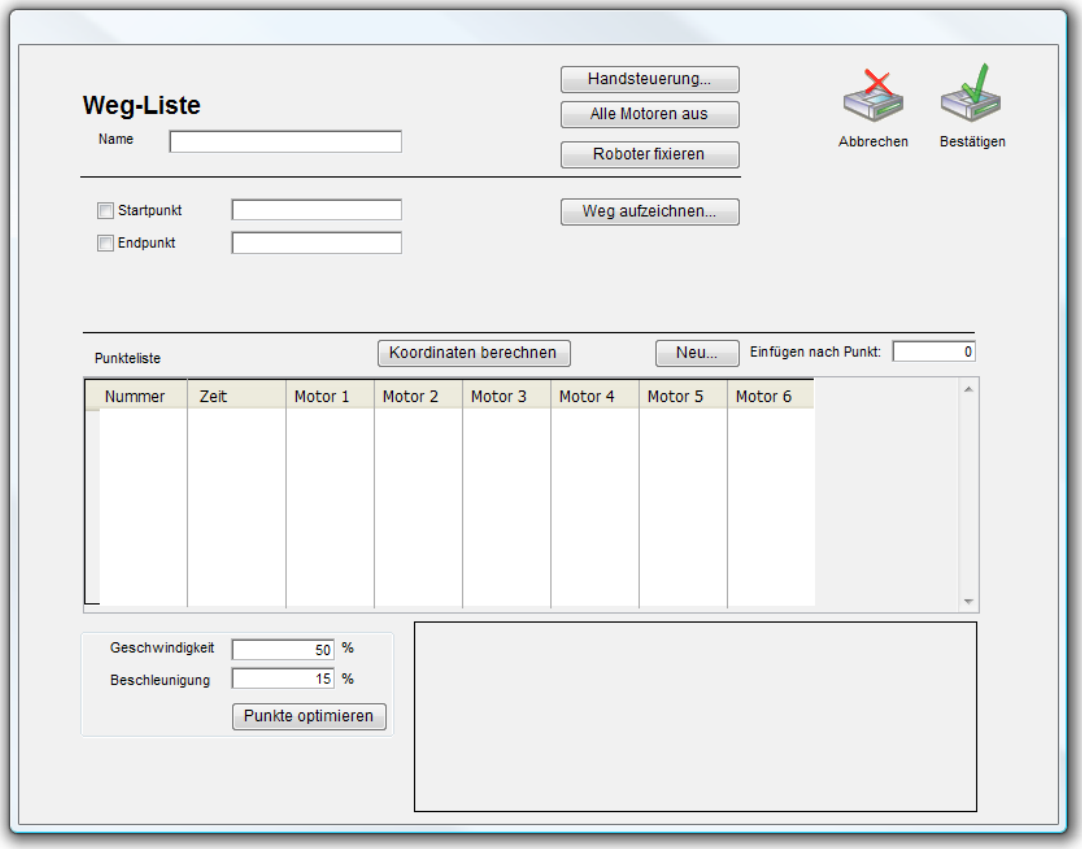
Mit Neu.. wird ein neuer Weg erstellt.

Der Name des Weges wird beim Erstellen festgelegt und kann anschliessend nicht mehr geändert werden. Eine Weg-Aufzeichnung basiert auf einem Start- und Endpunkt. Vom Startpunkt fährt der Roboter über dem aufgezeichneten Weg zum Endpunkt.

Es ist nicht notwendig, einen Start- und/oder Endpunkt zu definieren. Falls kein Startpunkt definiert ist, wird der Weg von der aktuellen Position gestartet. Um einen Startpunkt zu verwenden, muss *Startpunkt* aktiviert sein.

Wenn ein vorhandener Punkt verwendet werden soll, kann über die Schaltfläche >> ein Fenster geöffnet werden, in dem der entsprechende Punkt ausgewählt werden kann. Falls noch keine Punkte definiert wurden, kann dies über Erstellen nachgeholt werden. Dabei wird die aktuelle Position als Punkt gespeichert und in der Punkte-Liste eingefügt.

Durch die Auswahl von einem Endpunkt wird der aufgezeichnete Weg bis zu diesem Punkt vervollständigt.

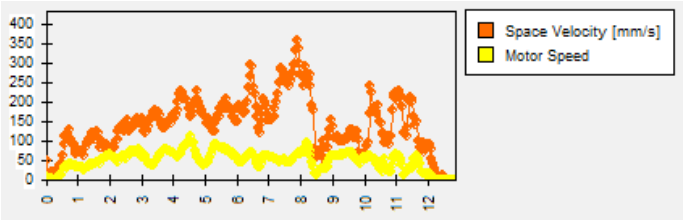


Weg aufzeichnen: Schritt für Schritt:

- Den Namen des Weges im Feld *Name* eingeben.
- Mittels >> den Start- und evtl. auch den Endpunkt aus der Punkteliste auswählen (optional).
- Weg aufzeichnen...
- Nach einer Meldung fährt der Roboter zum Startpunkt (falls einer angegeben wurde).
- Nach einer weiteren Meldung werden die Achsen ausgeschaltet und der gewünschte Weg kann aufgezeichnet werden.
- Durch das Drücken der Strg-Taste wird die Aufzeichnung beendet und die Achsen werden fixiert. Falls ein Endpunkt angegeben wurde, fahrt Katana zu diesem Punkt weiter.

Nach der Aufzeichnung erscheinen alle Zwischenpunkte, die auf dem Weg gespeichert wurden, in der Punkteliste. Es ist auch ersichtlich, wann ein Zwischenpunkt zeitlich (Millisekunden nach Start) aufgezeichnet wurde.

Über die Schaltfläche Koordinaten berechnen werden für jeden Wegpunkt die Koordinaten berechnet. Nachdem diese berechnet wurden, kann in der Grafik unten rechts die Geschwindigkeit des Tools abgelesen werden:

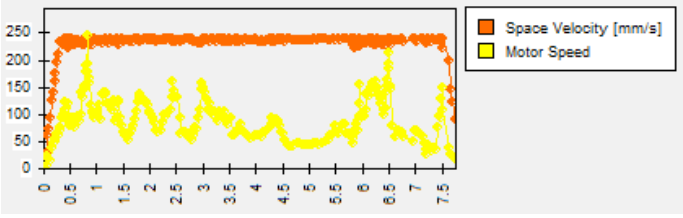


Mit einem Klick auf eine Zeile erscheinen rechts die drei bekannten Hilfsfunktionen Zwischenpunkt anfahren, Position speichern und Zwischenpunkt löschen. Auf der linken Seite kann über die Schaltfläche >> ein Fenster für diesen Wegpunkt geöffnet werden. In diesem können die Werte angepasst werden.

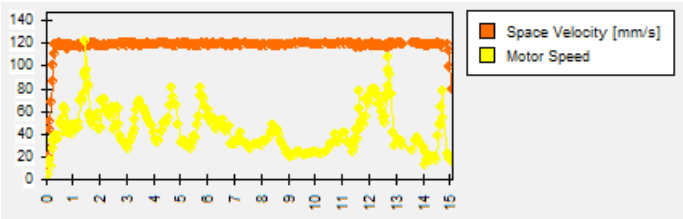
Mit Neu... wird ein Punkt nach der im Feld *Einfügen nach Punkt* angegebenen Nummer eingefügt. Falls eigene Punkte in die Liste aufgenommen wurden, ist es zu empfehlen, eine Optimierung der Punkte durchzuführen. Danach können die Punkte korrekt abgefahren werden.

In den meisten Fällen ist der aufgezeichnete Weg nicht optimal. Während der Bewegung von Hand werden viele kleine Pausen gemacht, die in einer Anwendung nicht erwünscht sind. Diese Pausen kommen überall vor, wo es für die gleiche Position mehrere Zwischenpunkte gibt. Mit Punkte optimieren werden alle unnötigen Punkte automatisch gelöscht. Für die Optimierung kann die gewünschte Geschwindigkeit und die Beschleunigung vorgegeben werden. Die Geschwindigkeit bezieht sich auf den Tool Center Point.

Mit einer Geschwindigkeit von 80% und einer Beschleunigung von 1 kann die Geschwindigkeit eines Weges beispielsweise so aussehen:



Der gleiche Weg sieht nach der Optimierung mit Geschwindigkeit 40% und Beschleunigung 1 so aus:



Weg importieren:

Es ist möglich einen Weg aus einer Textdatei zu importieren. Nachdem die Auswahl `Importieren' aktiviert wurde, kann über die Schaltfläche Weg importieren die gewünschte Datei geöffnet werden. Danach erscheinen die eingelesenen Punkte in der Liste.

Über die Schaltfläche Weg exportieren kann eine solche Datei erstellt werden.

Mit Bestätigen wird der Weg gespeichert und erscheint in der Liste.

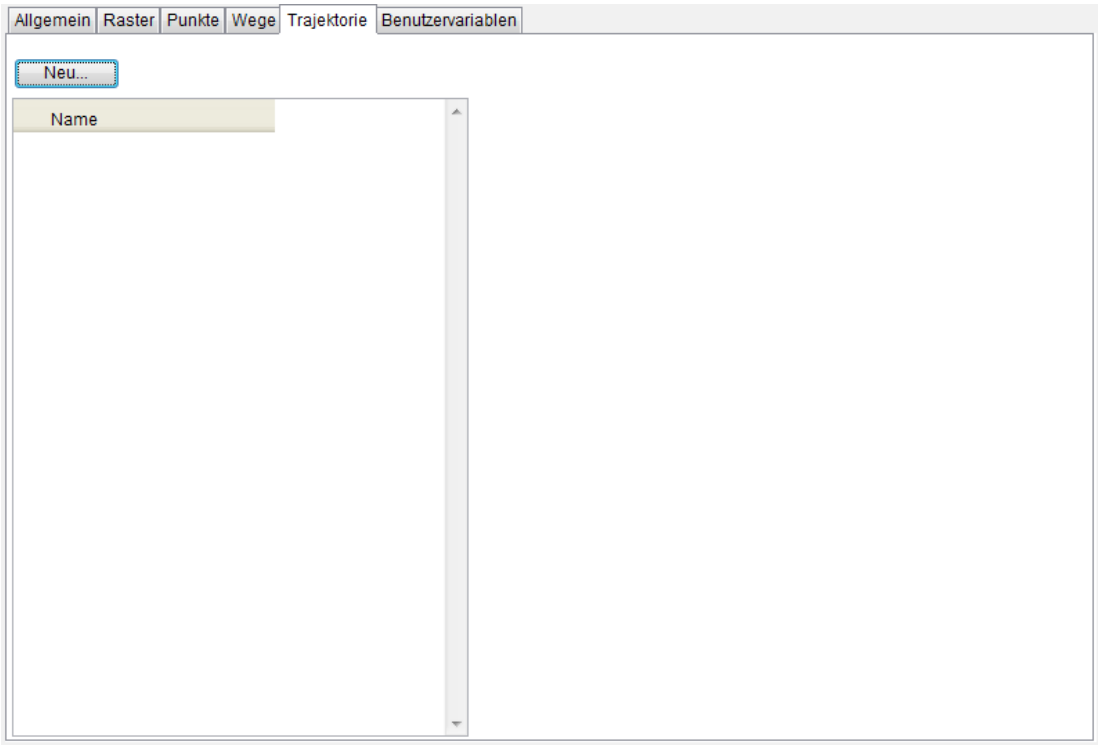
Nachdem ein Weg erstellt wurde, stehen in der Liste folgende Aktionen zur Verfügung:

- Weg, wie er aufgezeichnet wurde, abfahren.
- Weg rückwärts abfahren.
- Weg löschen.
- Weg duplizieren. Der duplizierte Weg wird mit dem Suffix “_New“ in die Liste eingefügt.

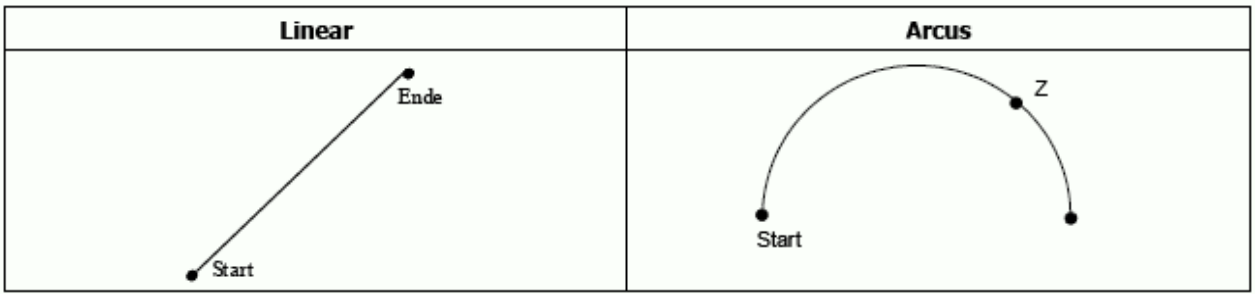
Hinweis

Im Projekt können Punkte (Rasterpunkte, Punkte im Raum, Punkte der Wege und Trajektorien) definiert werden. Verwendet werden sie erst im Programm, d.h. die Punkte werden dort über Fahrbefehle angefahren.

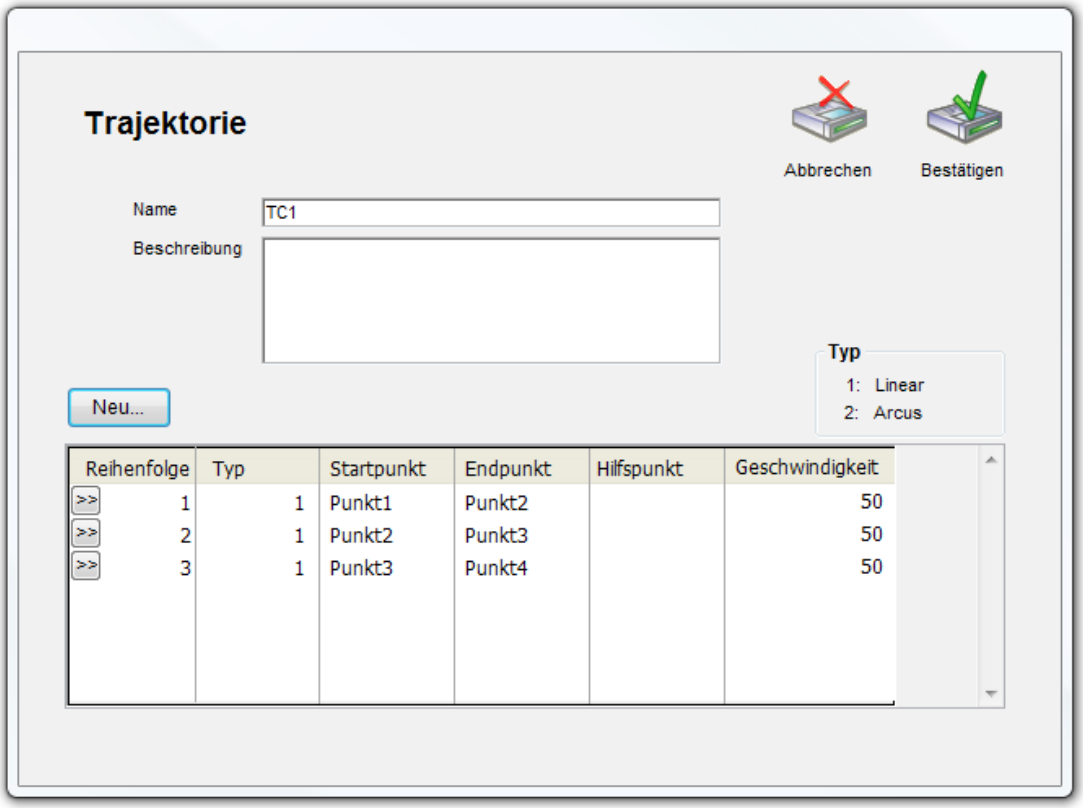
1.4.5 Trajektorie



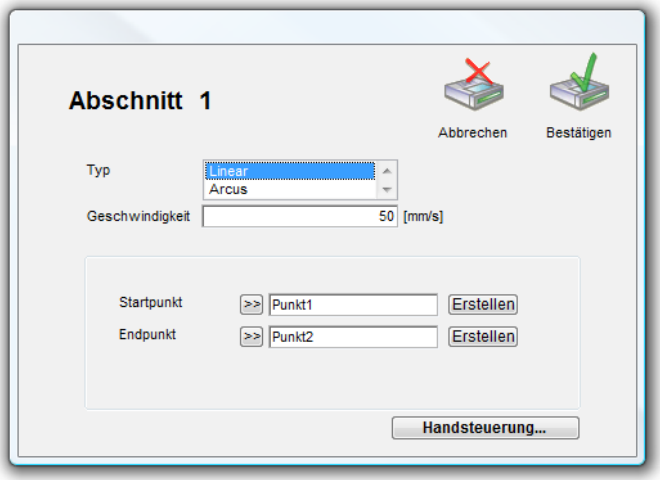
Eine Trajektorie ist eine Bahn, die aus Linearbewegungen (Linear) und Kreisbewegungen (Arcus) besteht. Die einzelnen Abschnitte werden aneinandergefügt und können schlussendlich als eine Bewegung – ohne Pause – ausgeführt werden. Eine Trajektorie kann aus beliebig vielen Abschnitten bestehen.



Mit Neu... wird eine neue Trajektorie erstellt.



Durch betätigen der Schaltfläche Neu... wird der Trajektorie ein neuer Abschnitt hinzugefügt. Es öffnet sich ein neues Fenster, in dem dieser Abschnitt definiert wird.



Bei Typ wird angegeben, ob es sich um eine Linear- oder Kreisbewegung handelt.

Bei einer Linearbewegung muss der Start und der Endpunkt definiert werden. Bei einer Kreisbewegung ein zusätzlicher Hilfspunkt (Zwischenpunkt auf der Kreisbahn).

Wenn ein vorhandener Punkt verwendet werden soll, kann über die Schaltfläche >> ein Fenster geöffnet werden. In diesem wird der entsprechende Punkt ausgewählt. Falls noch keine Punkte definiert wurden hat, kann dies mit Erstellen nachgeholt werden. Dabei wird die aktuelle Position als Punkt gespeichert und erscheint später in der Liste des Registers ***Punkte***.

Der Startpunkt eines Abschnitts muss mit dem Endpunkt des vorherigen Abschnitts übereinstimmen. Darum wird er automatisch eingefügt. Wäre dies nicht der Fall, könnte die Bewegung nicht ausgeführt werden.

Die Geschwindigkeit bezieht sich nur auf einen einzelnen Abschnitt. Ein zu grosser Geschwindigkeitsunterschied zwischen zwei Abschnitten kann eine unsaubere Bewegung zur Folge haben.

Mit Bestätigen wird das Fenster geschlossen und der Abschnitt erscheint in der Liste.

In der Liste auf der linken Seite kann über die Schaltfläche >> der Abschnitt geöffnet werden.

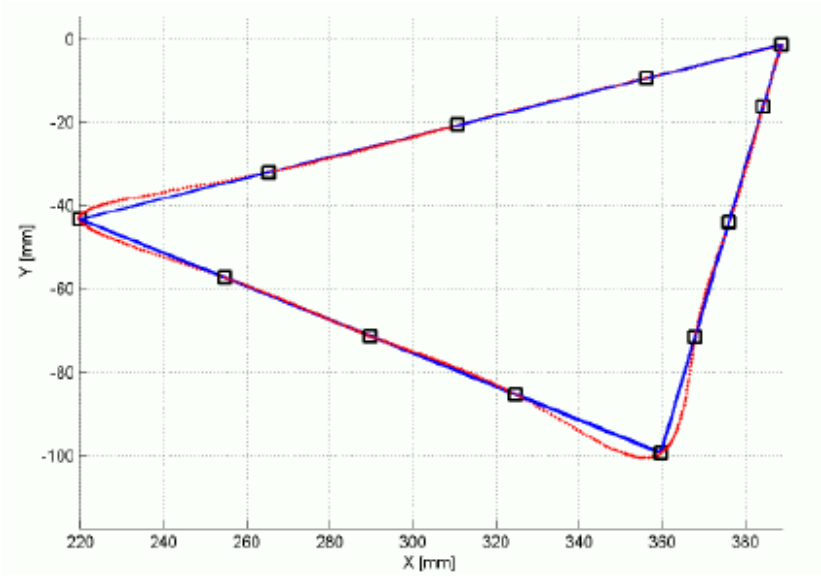
Nachdem ein Abschnitt auf der Liste ausgewählt wurde, erscheint rechts von den Werten das Symbol . Durch dessen Betätigung kann ein Abschnitt gelöscht werden.

Nach dem Löschen des Abschnittes muss überprüft werden, ob der Endpunkt des vorherigen Abschnitts mit dem Startpunkt des neuen Abschnitts übereinstimmt. Ist dies nicht der Fall, kann die Bewegung nicht ausgeführt werden.

Hinweis

Eine zusammengesetzte Bewegung wird in spitzen Übergangspunkten geglättet, weil sonst nicht mit der gesetzten Geschwindigkeit gearbeitet werden könnte. Auf der folgenden Grafik ist in blau die definierte und in rot die tatsächliche Strecke abgebildet. Die Bahn startet und endet in der Ecke oben rechts (nicht geglättet). In den Übergangspunkten ist zu erkennen, dass die Bahn um die Ecken überlagert wird.

Ist es erforderlich, dass die Punkte exakt angefahren werden, kann keine Trajektorie verwendet



werden. Die Abschnitte müssen einzeln im Programm abgefahren werden.

Mit Bestätigen wird der Weg gespeichert, so dass er in der Liste erscheint.

Trajektorie vorwärts abfahren.

Trajektorie rückwärts abfahren.

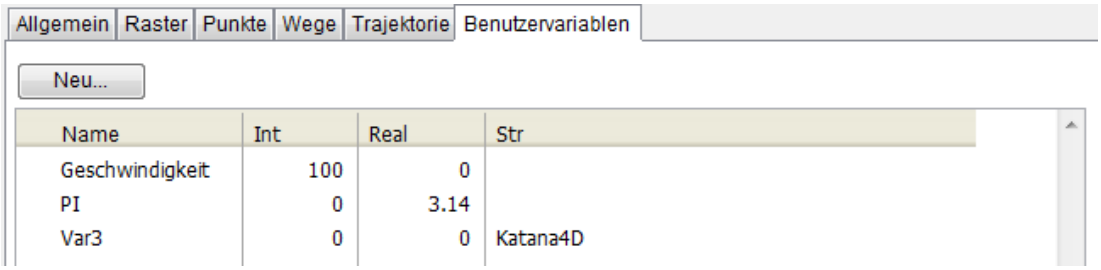
Trajektorie löschen.

Trajektorie duplizieren. Die duplizierte Trajektorie wird mit dem Suffix "_New" in die Liste eingefügt.

Hinweis

Im Projekt können Punkte (Rasterpunkte, Punkte im Raum, Punkte der Wege und Trajektorien) definiert werden. Verwendet werden sie erst im Programm, d.h. die Punkte werden dort über Fahrbefehle angefahren.

1.4.6 Benutzervariablen



Die Variablen eines Programms verlieren ihren Wert nach dem Programmende. In manchen Fällen ist es aber wünschenswert, dass der Wert einer Variablen auch nach dem Beenden von Katana4D erhalten bleibt. Dazu dienen Benutzervariablen. Diese sind sowohl auf Projekt- wie auch auf Programmebene verfügbar und werden persistent gespeichert. Durch den Gebrauch von Benutzervariablen kann verhindert werden, dass Variablen vor deren Verwendung initialisiert werden müssen.

Beispiel:

- Verwendung einer Variablen des Programms

  VarInt1 := 80

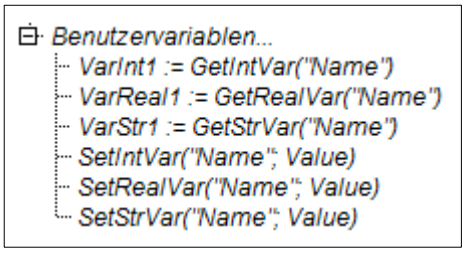
  MoveToPosLin (“Pos1“; VarInt1)

- Verwendung einer Benutzervariablen

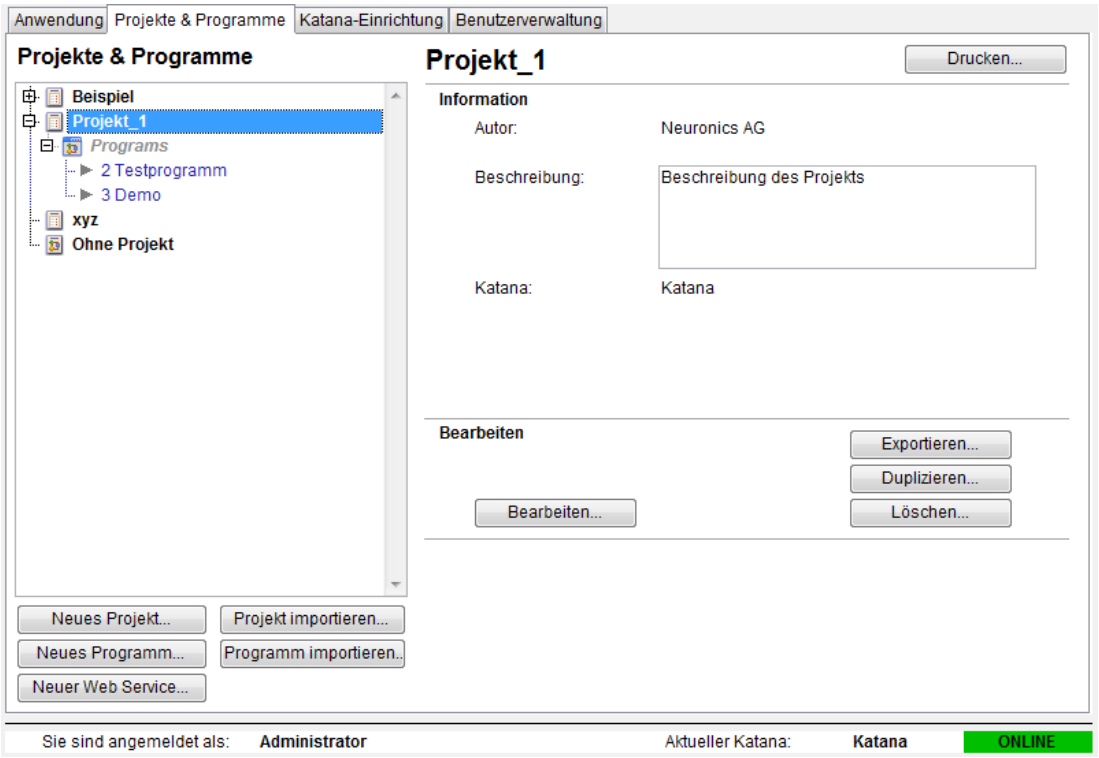
  MoveToPosLin (“Pos1“; GetIntVar (“Geschwindigkeit“))

Mit Neu... wird eine Benutzervariable erstellt und in die Liste eingefügt. Sie kann vom Typ Integer, Real oder String sein.

Im Programm können die Variablen gelesen und gespeichert werden. Die Beschreibung der dazu verwendeten Befehle ist in der Befehlsbeschreibung zu finden:



1.4.7 Projekt-Funktionen

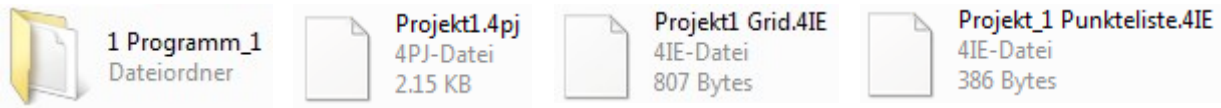


- Bearbeiten...

  Das Projekt-Fenster wird für die Bearbeitung geöffnet. Falls die Checkbox *mit dem Assistenten* (nur zu sehen, wenn das Projekt mit dem Assistenten erstellt wurde) aktiviert ist, wird das Projekt im Assistenten-Modus bearbeitet.

- Exportieren...

  Das Projekt wird zusammen mit allen Definitionen gespeichert. Zuerst muss der Speicherort angegeben werden. Danach wird ein Ordner mit dem Namen des Projekts erstellt. Die einzelnen Definitionen (Raster, Punkte, Wege und Trajektorien) werden als separate Dateien mit der Endung “4IE“ gespeichert. Das Projekt hat die Dateiendung “4pj“. Wenn keine Definitionen gemacht wurden, werden auch keine Dateien erstellt. Für jedes exportierte Programm wird ein zusätzlicher Ordner erstellt, der das jeweilige Programm beinhaltet.



- Duplizieren...

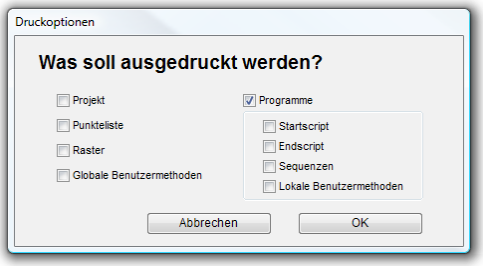
  Das Projekt, inklusive aller Definitionen, wird unter einem anderen Namen gespeichert.

- Löschen...

  Das Projekt wird nach der Bestätigung einer Warnung aus der Datenbank entfernt.

- Drucken...

  Es erscheint ein Dialog, in dem ausgewählt werden kann, was gedruckt werden soll.



Mit der Auswahl von *Programme* beziehen sich die eingerahmten Optionen auf alle Programme des Projekts, d.h es wird von jedem Programm das gewählte ausgedruckt.

- Neues Projekt...

  Ein neues Projekt kann erstellt werden.

- Neues Programm...

  Ein neues Programm kann erstellt werden.

- Neuer Web Service...

  Ein neuer Web Service kann erstellt werden.

- Projekt Importieren...

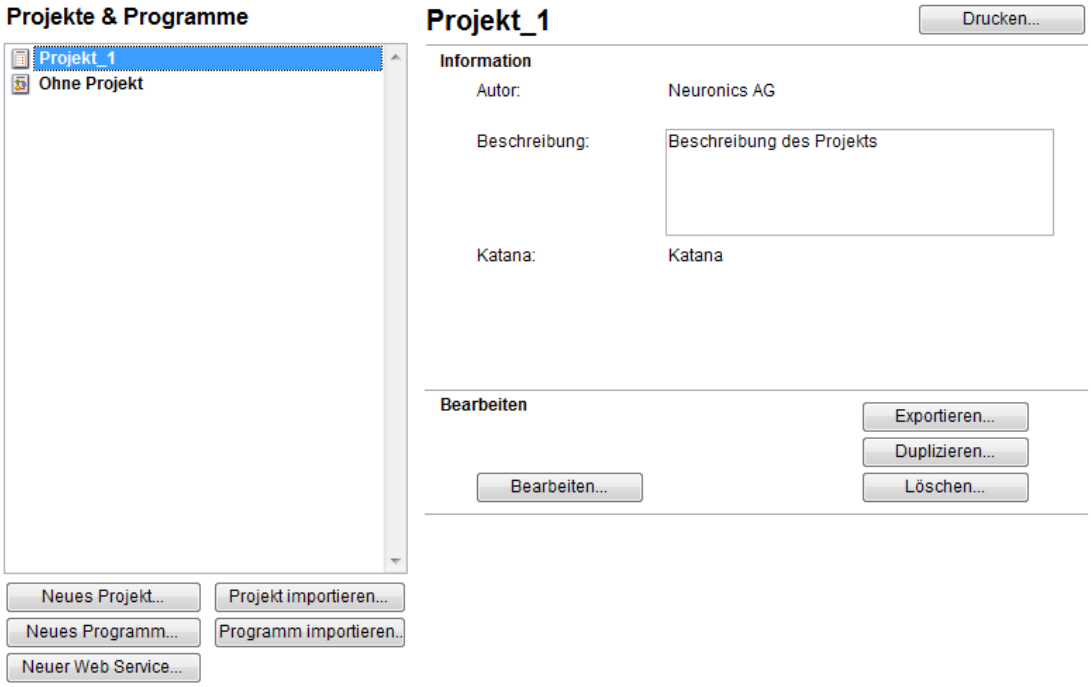
  Ein Projekt wird von einem Datenträger in die Datenbank importiert. Ist bereits ein Projekt mit dem gleichen Namen vorhanden, kann entschieden werden, ob das Projekt überschrieben oder mit einem neuen Namen importiert werden soll. Aus dem Ordner muss die Datei mit der Endung “4pj“ ausgewählt werden. Nach dem Importieren muss darauf geachtet werden, dass Raster allenfalls neu berechnet werden müssen.

- Programm Importieren...

  Ein Programm wird von einem Datenträger in die Datenbank importiert. Ist bereits ein Programm mit dem gleichen Namen vorhanden, kann entschieden werden, ob das Programm überschrieben oder mit einer neuen Nummer versehen werden soll. Aus dem Ordner muss die Datei mit der Endung “4pg“ ausgewählt werden.

1.5 Programm

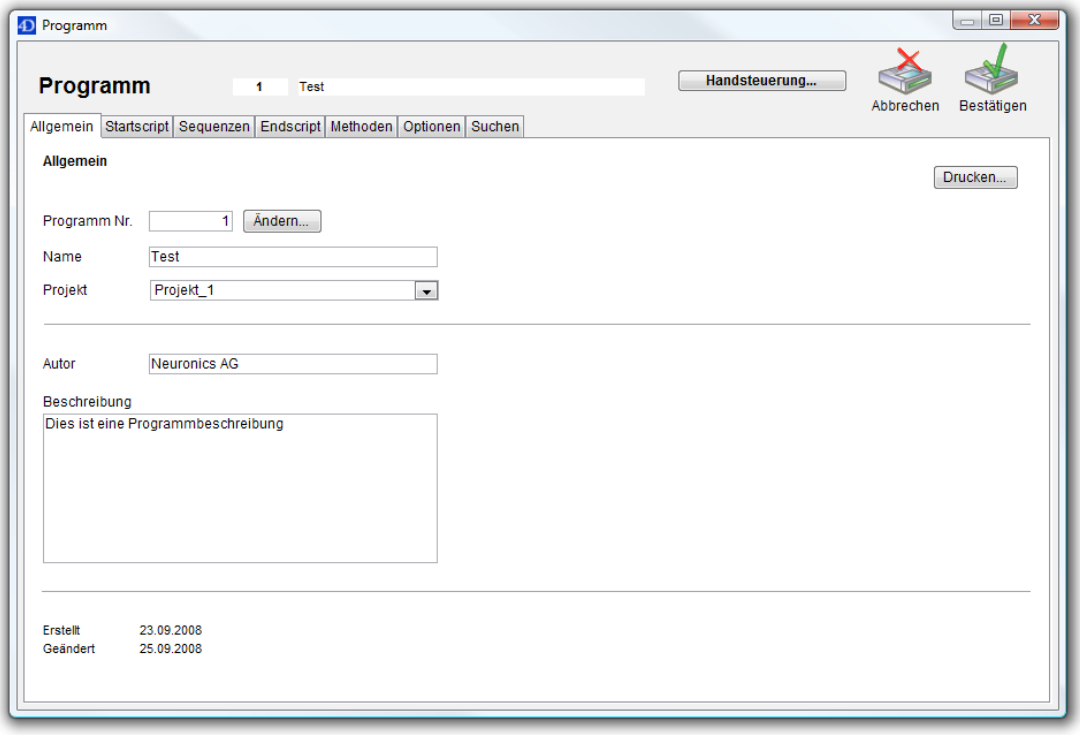
Ein Ablauf wird in einem Programm erstellt. Ein Programm ist im wesentlichen ein Script. Katana4D-Befehle und Definitionen eines Projekts (falls das Programm zu einem Projekt gehört) können verwendet werden.



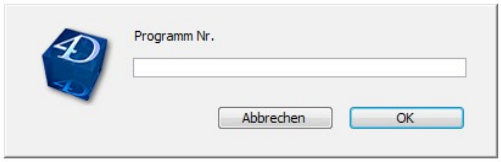
Durch Betätigen der Schaltfläche Neues Programm..., wird das Fenster ***Programm*** geöffnet.

Ein Programm ist üblicherweise ein Bestandteil eines Projektes. Nach dem Erstellen eines Programms wird das Projekt ausgewählt, das in der Liste aktiv ist. Es gibt auch die Möglichkeit, ein Programm zu erstellen, das zu keinem Projekt gehört. Dazu muss auf der Startseite des Programms, unter *Projekt*, der leere Eintrag auswählt werden. Diese Programme werden in der Liste unter 'Ohne Projekt' aufgeführt. Weil solche Programme kein Projekt verwenden, können sie auf keinen Katana zugreifen. Befehle mit dieser Voraussetzung funktionieren nicht.

1.5.1 Allgemein



Das Fenster ***Programm*** öffnet sich. Im Feld *Programm Nr.* wird automatisch eine Nummer eingetragen, die das Programm eindeutig identifiziert. Diese Programm-Nummer kann mit Ändern... geändert werden.



Mit Ändern... erscheint ein Fenster, in dem die neue Nummer eingegeben werden kann.

Die Programm-Nummer darf noch nicht vergeben sein. Sollte das der Fall sein, erscheint eine Fehlermeldung und die Eingabe wird abgebrochen.

Das Feld *Name* muss ausgefüllt werden. Die Felder *Autor* und *Beschreibung* (beide optional) sind nützlich, um Programm-Informationen zu hinterlegen. Im Feld *Projekt* werden alle Projekte aufgelistet, die vorhanden sind und zusätzlich ein leerer Eintrag. Gibt es bereits ein Projekt, das nützliche Definitionen enthält, sollte dieses ausgewählt werden. Die leere Zeile bedeutet, dass das Programm keinem Projekt zugeordnet wird.

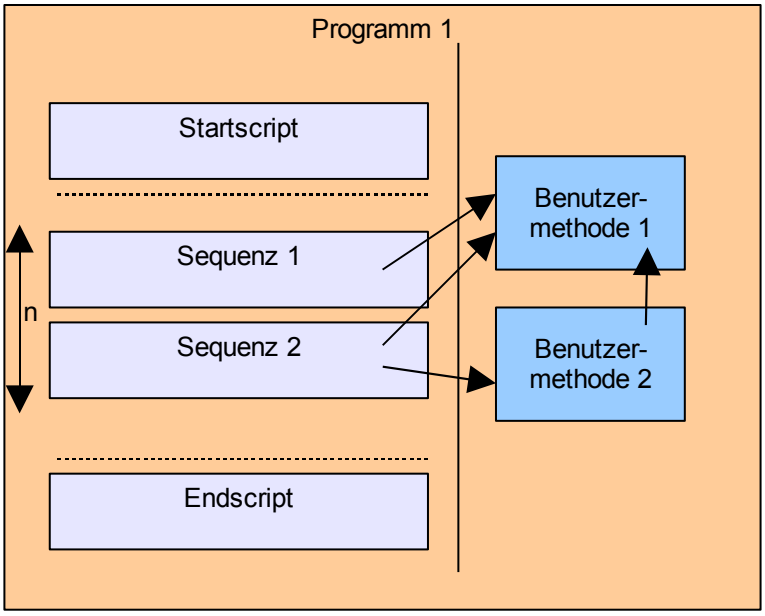
1.5.2 Aufbau des Programms

Ein Programm besteht aus vier verschiedenen Scripts:

- Startscript
- Sequenzen
- Endscript
- Methoden

Das Startscript wird einmal, ganz am Anfang der Ausführung eines Programms, abgearbeitet. Danach werden die einzelnen Sequenzen der Reihe nach ausgeführt. Bevor ein Programm ausgeführt wird, kann entschieden werden, wie viele Male die Sequenzen wiederholt werden sollen. Mit 0 werden die Sequenzen endlos wiederholt. Am Schluss wird das Endscript ausgeführt (kann im Register ***Optionen*** geändert werden).

Benutzermethoden werden nicht automatisch ausgeführt. Sie müssen vom Benutzer in einem Script aufgerufen werden. Benutzermethoden können andere Benutzermethoden aufrufen.



1.5.3 Abbruch des Programms

Ein Programm kann auf sechs verschiedene Arten abgebrochen bzw. beendet werden:

- Automatisch, nachdem die Sequenzen so viele Male, wie im Feld *Anzahl der Wiederholungen der Sequenzen* angegeben sind, abgearbeitet wurden. Dies ist der Normalfall. Bei dieser Variante wird das Endscript ausgeführt, sofern die Einstellungen im Register ***Optionen*** nicht verändert wurden.
- Durch Drücken der Tastatur-Kombination Strg-B. Die aktuelle Bewegung wird noch fertig ausgeführt und das Programm wird vorzeitig abgebrochen. Das Endscript wird ausgeführt, falls die Checkbox *Endscript immer ausführen* im Register ***Optionen*** angekreuzt ist.
- Durch die Software, wenn die Katana Variablen NOTAUS oder SaveAbort im Script auf 'True' gesetzt werden oder wenn 'ExitProgram' von einem anderen Programm ausgeführt wird.
  NOTAUS := TRUE: Die aktuelle Bewegung wird sofort gestoppt und das Programm vorzeitig beendet.
  SaveAbort := TRUE: Die aktuelle Bewegung wird fertig ausgeführt und das Programm vorzeitig beendet.
  ExitProgram (1): Beendet Programm 1.
- Durch die Hardware, wenn ein Kommunikationsfehler auftritt, z.B. wenn die Kommunikation zwischen Katana4D und dem Roboter unterbrochen wird. Bei dieser Variante wird das Endscript nicht ausgeführt.
- Durch eine Kollision. Beim Auftreten einer Kollision erscheint eine entsprechende Meldung. Wird diese mit “Ja“ bestätigt, wird das Programm beendet. Das Endscript wird nicht ausgeführt.
- Durch Drücken der Schaltfläche Beenden. Diese Schaltfläche erscheint, anstelle von Ausführen, sobald das Programm gestartet wurde. Das Endscript wird ausgeführt, sofern die Einstellungen im Register ***Optionen*** nicht verändert wurden.

1.5.4 Katana4D Script

In Katana4D wird unter einem Script ein Feld, das aus einer oder mehreren Anweisungen besteht, verstanden. Die Anzahl der Zeilen ist unbeschränkt. Pro Zeile kann eine, oder auch keine, Anweisung stehen.

Eine Anweisung ist:

- ein Kommentar
- ein Befehl aus der Befehlsbeschreibung
- eine Variablenzuweisung

1.5.4.1 Kommentare in Katana4D

Kommentare dienen der Beschreibung des Codes im Script. Sie werden mit dem Zeichen "`" eingeleitet. Dahinter kann ein beliebiger Text stehen. Alternativ kann ein Kommentar auch mit "//" eingeleitet werden.

Kommentierte Zeilen werden vom Programm nicht behandelt, sie werden ignoriert und übersprungen.

1.5.4.2 Katana-Befehle

Unter jedem Script befindet sich die Liste "Katana-Befehle".

Es gibt acht Kategorien von Befehlen:

- Aktionsbefehle
- Zustandsabfragen
- Schnittstellen
- Programmablauf-Steuerung
- Mathematische Funktionen
- Datum & Zeit
- String
- Array

Mit einem Klick auf das "Plus-Zeichen" öffnet sich die entsprechende Kategorie. Ein Befehl wird mit einem Doppelklick im Script eingefügt. Ein Befehl ist eine Anweisung, die aus mehreren Parametern bestehen kann. Werden als Parameter Definitionen eines Projekts, z.B. Punkte, verwendet, erscheint eine Auswahl. Durch einen Doppelklick wird die Auswahl als Parameter eingefügt. Im Kapitel "Befehlsbeschreibung" werden die einzelnen Befehle mit den dazugehörigen Parametern beschrieben. Die Katana-Befehle werden fortlaufend ergänzt.

1.5.4.3 Variablen

Eine Variable ist ein Wertspeicher mit einem bestimmten Datentyp. In der Liste "Variablen" wird zuerst die Kategorie mit dem "Plus-Zeichen" geöffnet. Mit einem Doppelklick auf eine Zeile, wird die entsprechende Variable im Script eingefügt. Mit ":=" wird einer Variable ein Wert zugewiesen. Nach der Zuweisung kann die Variable in jedem Script verwendet werden. Im Gegensatz zu den Benutzervariablen bleibt ihr Wert nach Programmende nicht erhalten.

Beispiel

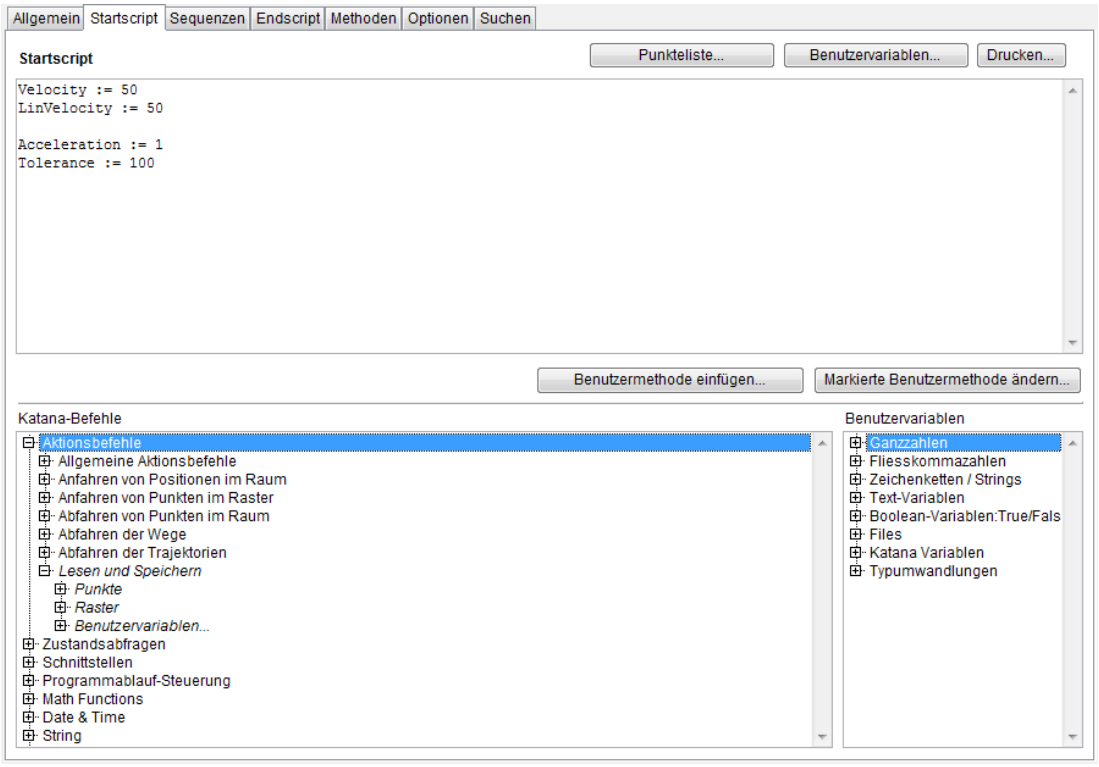
VarInt1 := 60

MoveAxis (2; VarInt1) `Bewegt Achse 2 zur Position VarInt1

Variablen sind besonders dann nützlich, wenn ein Wert mehrmals im Programm gebraucht wird und dieser später geändert werden muss. Würde man nicht mit Variablen arbeiten, müsste man den Wert überall ändern, wo er vorkommt.

Integer (VarInt1-10)	In diesen Variablen können nur ganze Zahlen abgespeichert werden. Kommastellen werden keine akzeptiert. Werte zwischen -2^{31} (inklusive) und $+2^{31}$ (exklusive) sind erlaubt. Bsp. VarInt1 := 54
Real (VarReal1-10)	In diesen Variablen können Zahlen mit Kommastellen abgespeichert werden. Werte zwischen -2^{31} (inklusive) und $+2^{31}$ (exklusive), mit maximal 15 Nachkommastellen, sind erlaubt. Bsp. VarReal1 := 54.587
String (VarStr1-10)	In Strings können Zeichenketten mit bis zu 255 Zeichen abgespeichert werden. Der Inhalt muss zwischen Anführungs- und Schlusszeichen stehen. Bsp. VarStr1 := “abc-123-*%$“
Text (VarText1-10)	Gleich wie eine String-Variable, kann aber bis zu 32'000 Zeichen enthalten.
Boolean (VarBool1-10)	In diesen Variablen können Wahrheitswerte, d.h. 'True' oder 'False', abgespeichert werden. Bsp. VarBool1 := False
Files (VarFile1-10)	In diesen Variablen können Dateien gespeichert werden. Der Wert wird mit Hilfe von Open document (Dokument; Dateityp) festgelegt. Bsp. VarFile1 := Open document(“C:\test”; ”TXT”)
Katana Variablen (Boolean)	**SaveAbort := TRUE** Bricht das Programm ab und führt die Benutzermethode 'OnSaveAbort' aus, sofern sie vom Benutzer erstellt wurde. **NOTAUS := TRUE** Wird diese Variable gesetzt, stoppt Katana sofort und das Programm wird abgebrochen. Eine Meldung, dass ein Notaus aufgetreten ist, wird abgesetzt. **<>SaveAbortAll und <>NotAusAll** Diese Variablen sind global. Sie werden verwendet, falls mehrere Katanas im Betrieb sind und gestoppt werden müssen.
Interprozess-Variablen	In Katana4D gibt es die Möglichkeit, mehrere Programme gleichzeitig auszuführen. Die Kommunikation zwischen den Programmen kann mit Hilfe von Schnittstellen (siehe Abschnitt Schnittstellen) oder auch, was schneller und einfacher ist, mit Hilfe der Interprozess-Variablen erfolgen. Ein ausgeführtes Programm beansprucht einen Prozess, die Interprozess-Variablen sind in allen Prozessen sichtbar, d.h. eine solche Variable kann in allen Programmen gelesen und geschrieben werden. Interprozess-Variablen bestehen aus der Zeichenfolge “<>“ gefolgt von einem Bezeichner. Bsp: *<>VarInt1*

1.5.5 Startscript und Endscript



Das Start- und Endscript sieht gleich aus. Es wird am Anfang bzw. am Ende des Programms ausgeführt.

Im Startscript werden üblicherweise Variablen initialisiert, d.h. sie werden auf einen bestimmten Anfangswert gesetzt. Zudem können Bewegungsbefehle ausgeführt werden, z.B. wenn der Roboter am Anfang an eine Startposition fahren soll, bevor die einzelnen Sequenzen abgearbeitet werden.

Beispiele

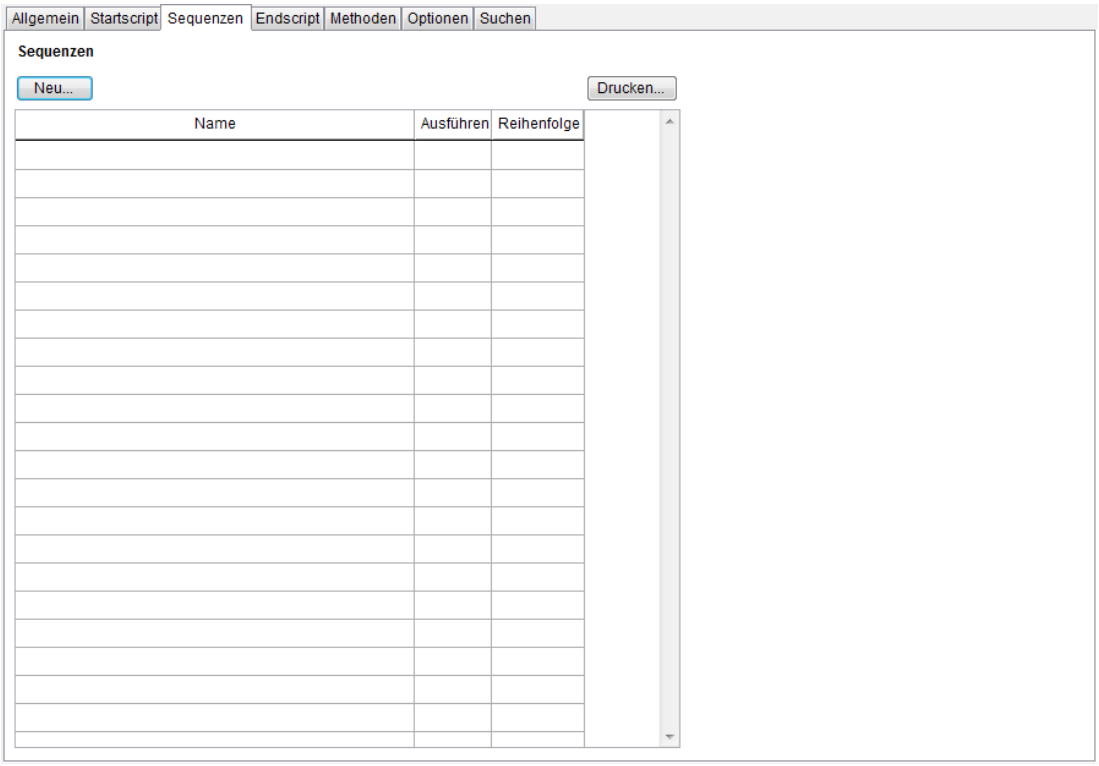
VarInt1 := 60 `Variablen-Initialisierung

MoveToPoint(“StartPosition“;80; 1) `Punkt “StartPosition“ anfahren

Neben den beschriebenen Katana-Variablen, gibt es noch vier weitere vordefinierte Variablen *Velocity*, *LinVelocity*, *Acceleration* und *Tolerance.* Diese werden in den meisten Bewegungs-befehlen verwendet. Darum werden sie standardmässig im Startscript definiert.

Das Endscript ist besonders nützlich, wenn der Roboter nach der Beendigung der Sequenzen in eine sichere Position gebracht werden muss. Im Endscript können ebenfalls alle Befehle verwendet werden.

1.5.6 Sequenzen

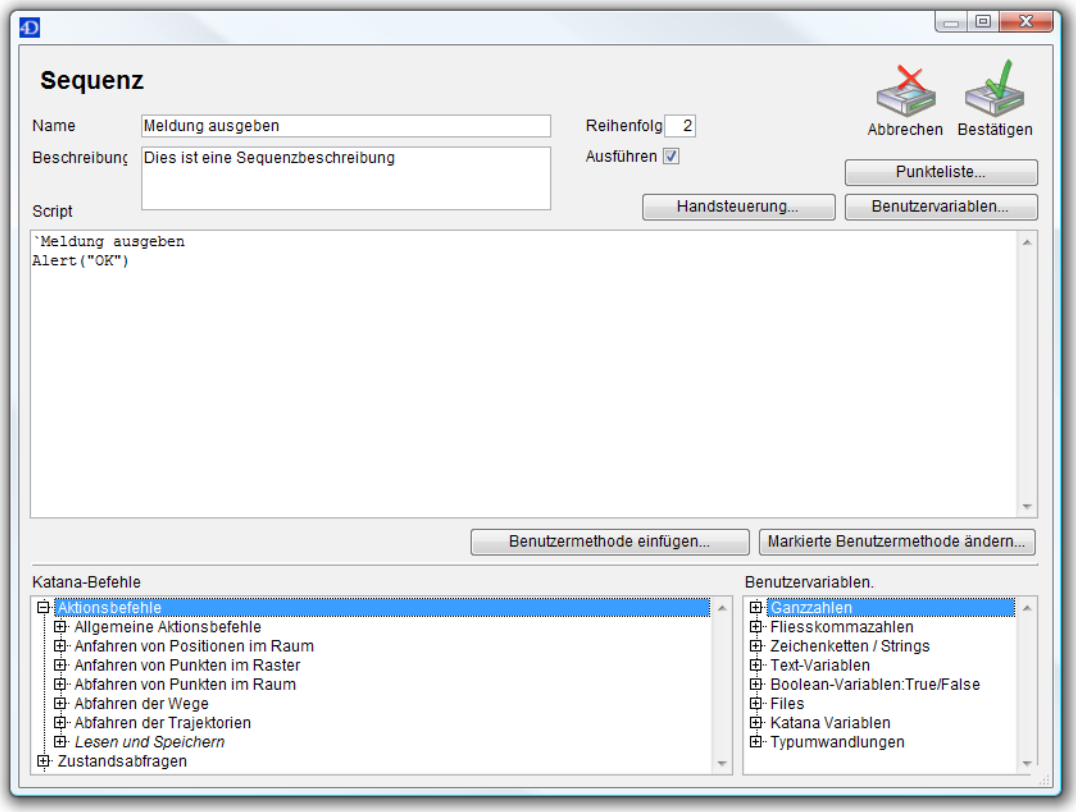


In einem Programm können beliebig viele Sequenzen erstellt werden. Sie werden zwischen Start- und Endscript der Reihe nach abgearbeitet. Das bedeutet, dass zwischen den einzelnen Sequenzen nicht hin und her gesprungen werden kann. Es ist empfehlenswert, den gesamten Ablauf in mehrere kleine Sequenzen zu unterteilen. Damit wird eine bessere Übersicht des Programms gewährleistet. Ausserdem wird die Fehlersuche und das Testen erleichtert, weil einzelne Sequenzen ein- bzw. ausgeschaltet werden können. Ihre Reihenfolge kann somit bedingt geändert werden.

Mit Neu... erscheint das Fenster zum Erstellen einer Sequenz.

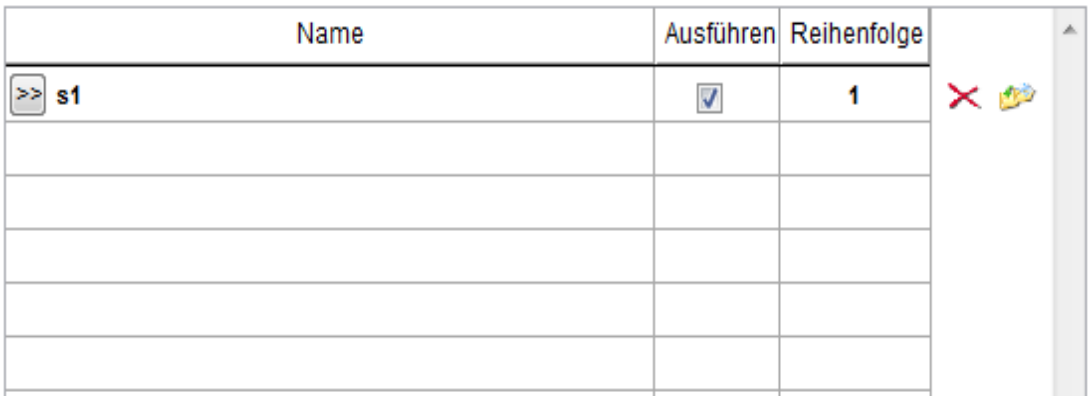
Im Feld *Name* wird der Titel der Sequenz eingetragen. Dieses Feld muss zwingend ausgefüllt werden. Der Name muss nicht aus einem Wort bestehen. Mehrere Wörter, wie z.B. “Greifer öffnen und warten auf Signal“, sind möglich. Im Feld *Beschreibung* (optional) kann die Sequenz beschrieben werden. Nach dem Speichern der Sequenz erscheint ihr Name in der Sequenzenliste.

Da Sequenzen automatisch ausgeführt werden, muss die Reihenfolge für die Abarbeitung definiert sein (Feld *Reihenfolge*). Die Sequenz wird nur ausgeführt, wenn ein Haken in der Checkbox *Ausführen* gesetzt ist.



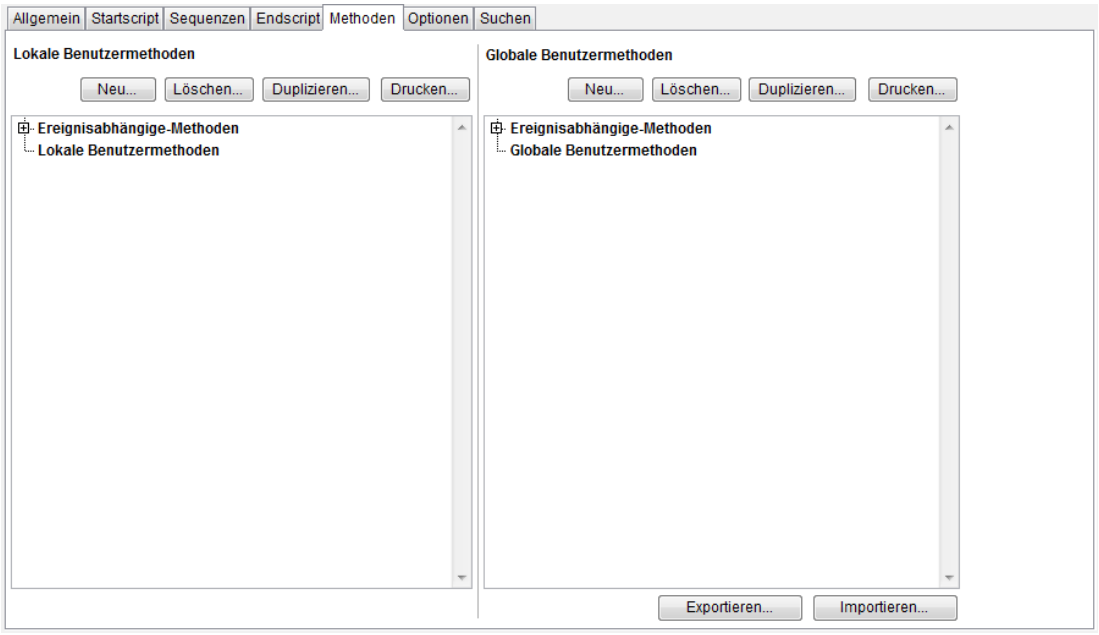
Das Script einer Sequenz unterscheidet sich nicht vom Startscript. Unten rechts befinden sich die Variablen, unten links die “Katana-Befehle“. Da die Punkte aus der Punkteliste des Projekts häufig verwendet werden, gibt es einen direkten Link zum Register ***Punkte***. Aus dem gleichen Grund gibt es auch die Hilfsfunktion Markierte Benutzermethode ändern....

Durch einen Klick auf Bestätigen schliesst sich das Fenster und die neu erstellte Sequenz erscheint in der Sequenzenliste.



Mit >> kann die ausgewählte Sequenz bearbeitet werden. Mit den zwei Bildfunktionen auf der rechten Seite kann die Sequenz gelöscht oder dupliziert werden.

1.5.7 Benutzermethoden



Benutzermethoden sind Scripts, die während dem Programmablauf nicht automatisch ausgeführt werden. Sie müssen vom Benutzer selbst oder über ein auftretendes Ereignis aufgerufen werden.

Benutzermethoden sind mit Funktonen einer konventionellen Programmiersprache vergleichbar. Sie können in jedem Script mit dem Befehl 'RunUserMethod (“NameDerMethode“)' oder direkt über den Namen aufgerufen werden.

1.5.7.1 Lokale Benutzermethoden

Lokale Benutzermethoden werden für ein Programm erstellt. Sie sind nur in diesem sicht- und ausführbar. Die Ausführung lokaler Benutzermethoden anderer Programme ist nicht möglich.

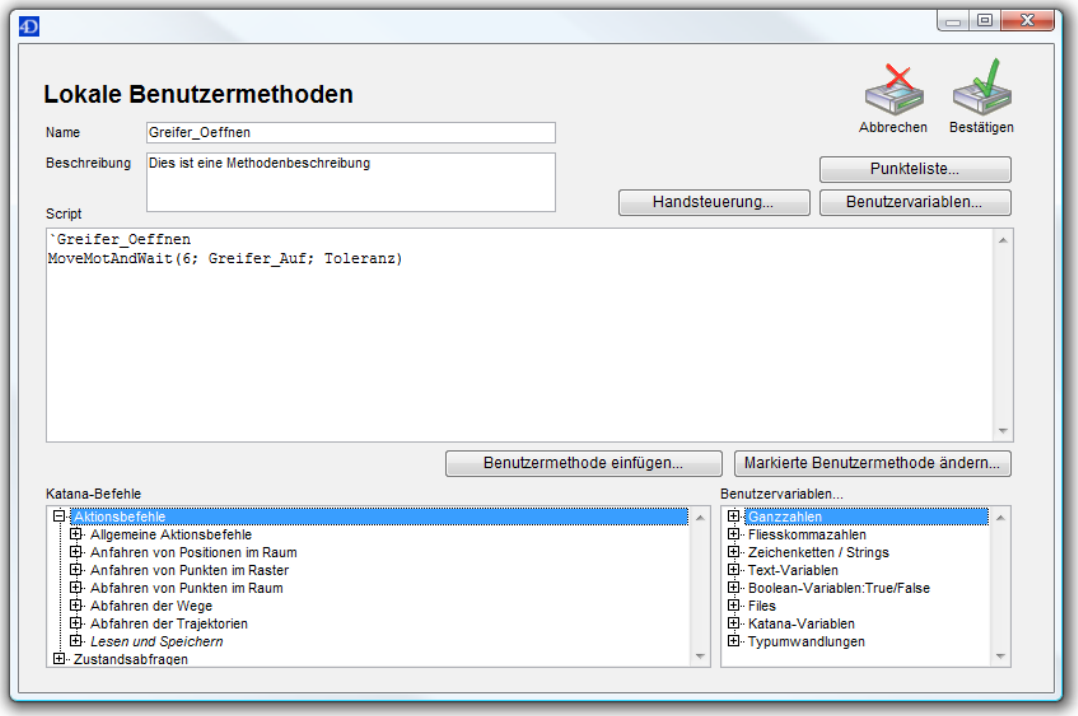
1.5.7.2 Globale Benutzermethoden

Globale Benutzermethoden werden in einem Programm erstellt und können von allen Programmen aufgerufen werden. Sie erscheinen in jedem Programm in der Liste der globalen Benutzermethoden.

Lokale Benutzermethoden haben höhere Priorität als globale. Wenn es eine lokale und eine globale Benutzermethode mit dem gleichen Namen gibt, wird die lokale ausgeführt.

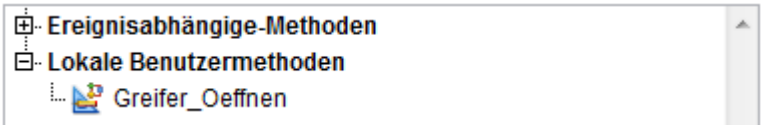
Benutzermethoden sollten, wenn möglich, lokal definiert werden. Eine globale Definition ist nur zu empfehlen, wenn andere Programme die Methode zwingend verwenden müssen.

Mit Neu... wird eine neue lokale bzw. globale Benutzermethode erstellt.



Das neu geöffnete Fenster ***Benutzermethoden*** sieht dem Fenster ***Sequenzen*** sehr ähnlich. Da Benutzermethoden vom Benutzer aufgerufen werden, gibt es keine Reihenfolge für deren Ausführung und keine Checkbox *Ausführen*.

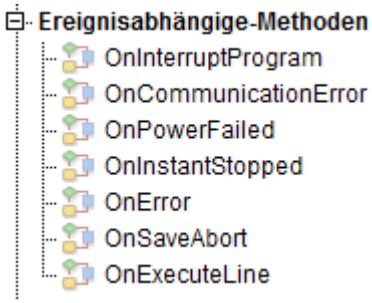
Nachdem die Benutzermethode gespeichert wurde, erscheint sie in der Liste.



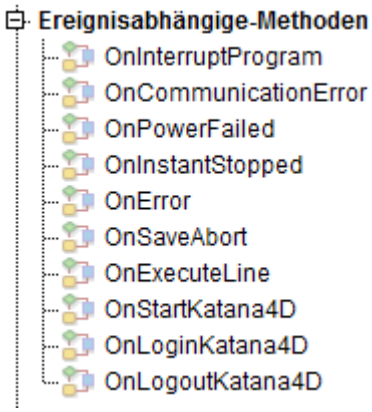
Die ausgewählte Benutzermethode kann mit einem Doppelklick bearbeitet, mit Löschen... gelöscht und mit Duplizieren... kopiert werden. Beim Duplizieren wird dem Namen eine 2 hinzugefügt, d.h. die Benutzermethode “Test“ wird in “Test2“ umbenannt. Der Name kann danach bearbeitet werden. Mit Drucken... können die Benutzermethoden, mit deren Inhalt, ausgedruckt werden.

1.5.7.3 Ereignisabhängige Benutzermethoden

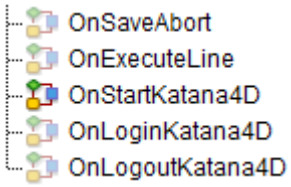
Lokal



Global



Eine ereignisabhängige Benutzermethode wird mit einem Doppelklick auf die entsprechende Methode erstellt. Ihr Code wird ausgeführt, wenn ein bestimmtes Ereignis (siehe Tabelle unten) aufgetreten ist. Nach der Erstellung wird das linke Symbol aktiviert. Mit Löschen... wird die Methode gelöscht und das Symbol wieder deaktiviert.



OnInterruptProgram	wird ausgeführt, nachdem der Benutzer das Programm mit der Tasten-Kombination Strg-B beendet hat. *
OnCommunicationError	wird ausgeführt, wenn das Programm aufgrund eines Kommunikationfehlers beendet wird. *
OnPowerFailed	wird ausgeführt, wenn der “Power Fail“-Eingang der Controlbox gesetzt wurde.
OnInstantStopped	wird ausgeführt, wenn der “Soft Stop“-Eingang der Controlbox gesetzt wurde.
OnError	wird bei jedem aufgetretenen Fehler ausgeführt. Damit kann auf Fehler reagiert werden. *
OnSaveAbort	wird ausgeführt, nachdem der Benutzer die Variable SaveAbort oder <>SaveAbortAll auf 'True' gesetzt hat. *
OnExecuteLine	wird ausgeführt, bevor ein normaler Befehl aus einem Script ausgeführt wird (nützlich für Notaus-Abfragen).
OnStartKatana4D	wird sofort ausgeführt, nachdem Katana4D gestartet wurde.
OnLoginKatana4D	wird ausgeführt, nachdem sich der Benutzer angemeldet hat.
OnLogoutKatana4D	wird ausgeführt, nachdem sich der Benutzer abgemeldet hat.

* Das laufende Programm wird beendet.

Ereignisabhängige Benutzermethoden unterscheiden sich in der internen Struktur nicht von normalen, lokalen oder globalen Benutzermethoden. Der Name ist fix und darf nicht geändert werden. Wird der Name trotzdem geändert und die Methode unter diesem Namen gespeichert, erzeugt Katana4D eine normale, globale Benutzermethode mit diesem Namen. Dementsprechend wird sie auch behandelt.

Die globalen Benutzermethoden können mit Exportieren... auf einem Datenträger gespeichert und mit Importieren... wieder importiert werden. Beim Exportieren wird die Datei “Globale Benutzermethoden.4IE“ erstellt. Die lokalen Benutzermethoden können nicht exportiert bzw. importiert werden. Sie werden übergeordnet als ganzes Programm behandelt.

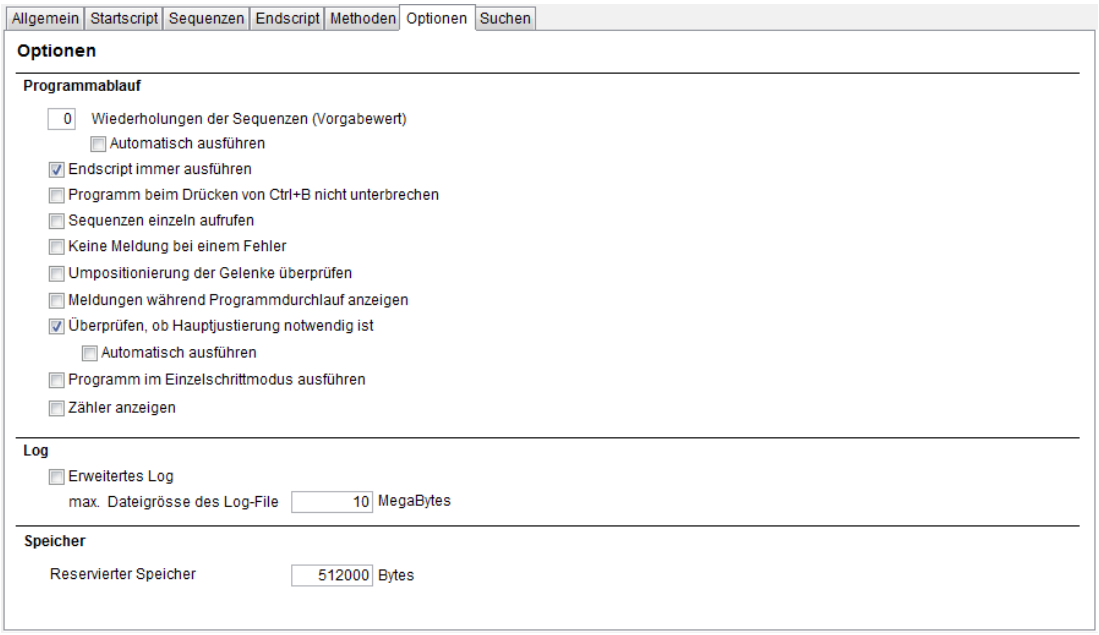
Mit Drucken... wird die Liste der Benutzermethoden, mit deren Inhalt, ausgedruckt.

In den Benutzermethoden 'OnStartKatana4D', 'OnLoginKatana4D' und 'OnLogoutKatana4D' darf nicht auf den Roboter zugegriffen werden. Falls es aber beim Start dennoch nötig ist, kann ein Programm erstellt werden, das die gewünschten Vorgänge ausführt. Dieses Programm kann in der Benutzermethode 'OnLoginKatana4D' über den Befehl 'RunProgram(Nr)' gestartet werden. Es gilt generell zu beachten, dass nur auf den Roboter zugegriffen werden kann, der dem Projekt zugeordnet ist.

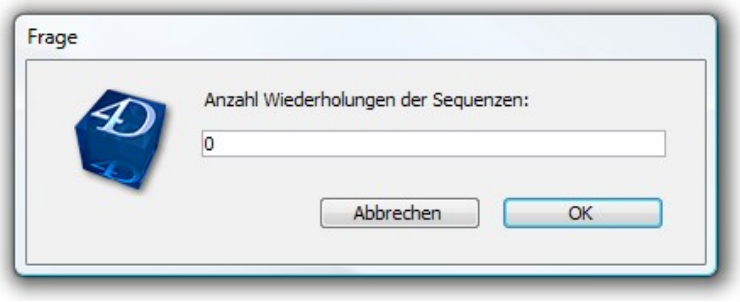
Hinweis

Ereignisabhängige Benutzermethoden, die beim Auftreten eines Fehlers aufgerufen werden, dürfen selbst keine weiteren Fehler verursachen. Dies hätte ansonsten eine Endlosschlaufe zur Folge. Wenn man z.B. einen Fahrbefehl in 'OnCommunicationError' verwendet, verursacht dieser einen weiteren Kommunikationsfehler.

1.5.8 Optionen



Vor der Ausführung eines Programms erscheint ein Fenster, in dem die Anzahl der Wiederholungen der Sequenzen angegeben werden kann.



Der Standardwert '0' bedeutet, dass die Sequenzen solange ausgeführt werden, bis der Benutzer das Programm mit der Kombination Ctrl-B vorzeitig abbricht oder die Schaltfläche Beenden klickt. Dieser Wert kann im Feld *Wiederholungen der Sequenzen* geändert werden.

Mit *Automatisch ausführen* wird das Programm ohne Benutzeraufforderung zur Eingabe der Anzahl der Wiederholungen automatisch ausgeführt.

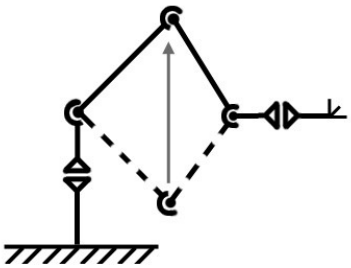
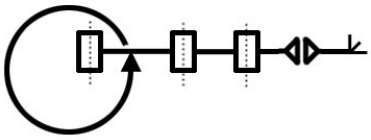
Falls ein Programm vorzeitig abgebrochen wird, wird das Endscript nur ausgeführt, wenn die Checkbox *Endscript immer ausführen* aktiviert ist.

Soll ein Programm nicht über die Kombination Ctrl-B abgebrochen werden können, kann die Option *Programm beim Drücken von Ctrl-B nicht unterbrechen* ausgewählt werden.

Mit *Sequenzen einzeln aufrufen* können die einzelnen Sequenzen einzeln aufgerufen werden.

Wenn *Keine Meldung bei einem Fehler* ausgewählt ist, wird das Programm nach einem Fehler ohne Meldung beendet. Es macht nur Sinn, die Meldung nicht anzuzeigen, wenn Sie die ereignisabhängige Benutzermethode 'OnError' verwenden und eine eigene Meldung ausgeben oder Katana4D extern gesteuert wird.

Falls die Option *Umpositionierung der Gelenke überprüfen* ausgewählt ist, wird bei einer Umpositionierung der Gelenke eine Warnmeldung angezeigt, bei der man das Programm abbrechen oder fortsetzen kann.

	
Umpositionierung der Gelenke 2,3 und 4	Umpositionierung von Gelenk 1

Ist die Checkbox *Meldungen während Programmdurchlauf anzeigen* angekreuzt, werden die Befehle, die gerade an den Roboter gesendet werden, auf dem Bildschirm angezeigt.

Über die Option *Überprüfen, ob Hauptjustierung notwendig ist* wird festgelegt, dass beim Start des Programms überprüft werden soll, ob eine Hauptjustierung notwendig ist. Ist dies der Fall, erscheint eine Meldung, dass der Roboter zurückgesetzt wird und eine Justierung notwendig ist. Nach der Bestätigung wird diese ausgeführt.

Mit *Automatisch ausführen* wird diese Meldung unterdrückt. Die Hauptjustierung wird direkt ausgeführt.

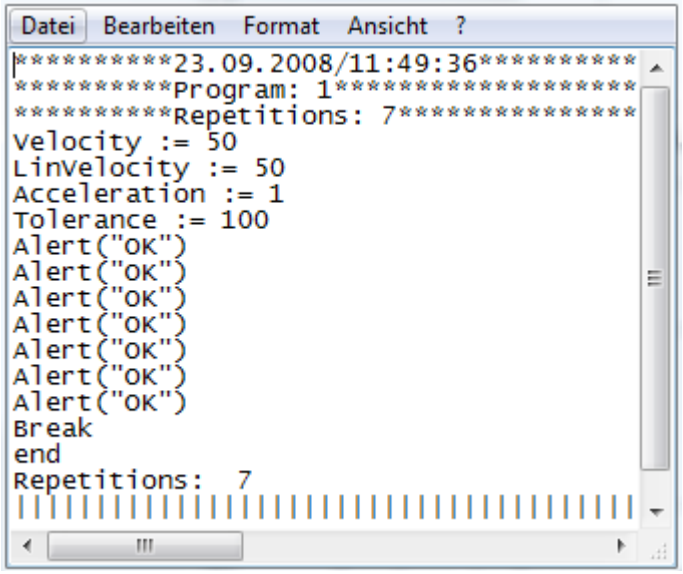
Mit *Programm im Einzelschrittmodus ausführen* erscheint, bevor eine Zeile ausgeführt wird, ein Fenster. In diesem muss die Ausführung der Zeile manuell bestätigt werden.

Mit der Option *Zähler anzeigen* können zusätzliche Informationen in einem Fenster angezeigt werden. Dazu gehört die Anzahl der Ausführungen, die Zykluszeit und die durchschnittliche Zykluszeit.

Standardmässig führt Katana4D im Hintergrund ein Log-Buch. Bei der Ausführung eines Programms wird auf der Festplatte, unter <Installationsverzeichnis>\Log, eine Text-Datei mit dem Namen *“log_<ProgrammNr>.txt“* (z.B. *“log_6.txt“*) erstellt. Darin werden folgende Informationen gespeichert:

- Datum und Uhrzeit beim Programmstart
- Programm-Nummer
- angegebene Anzahl der Wiederholungen
- alle Script-Befehle, die ausgeführt wurden
- die Anzahl der Wiederholungen bis Programmabbruch, falls frühzeitig abgebrochen wurde
- Grund für den Programmabbruch (z.B. Kollision)

Mit *Erweitertes Log* wird zusätzlich die Position des Roboters vor und nach einem Fahrbefehl in der Datei gespeichert. Dieser Zusatz ist mit Verzögerungen im Programmablauf verbunden. Aus diesem Grund sollte das erweiterte Log nur für Testzwecke verwendet werden.

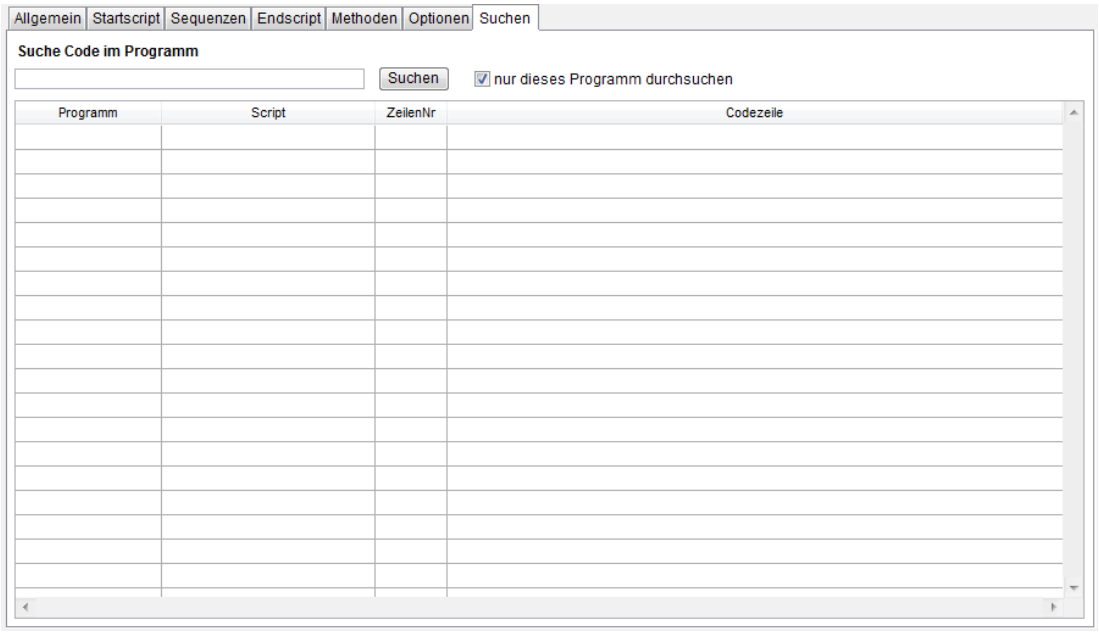


Im Feld *max. Dateigrösse des Log-File* kann angegeben werden, wie viel Speicher die Datei beanspruchen darf. Es gilt zu beachten, dass sich diese Einstellung nur auf das jeweilige Programm bezieht. Jedes Programm erstellt eine neue Datei mit separaten Einstellungen.

Reservierter Speicher legt fest, wie viel Speicher dem Programm maximal zur Verfügung steht. Der Standardwert beträgt 512'000 Bytes. Falls beim Ablauf eines Programms die Meldung erscheint, dass der Stapelspeicher nicht ausreicht, muss dieser Wert erhöht werden.

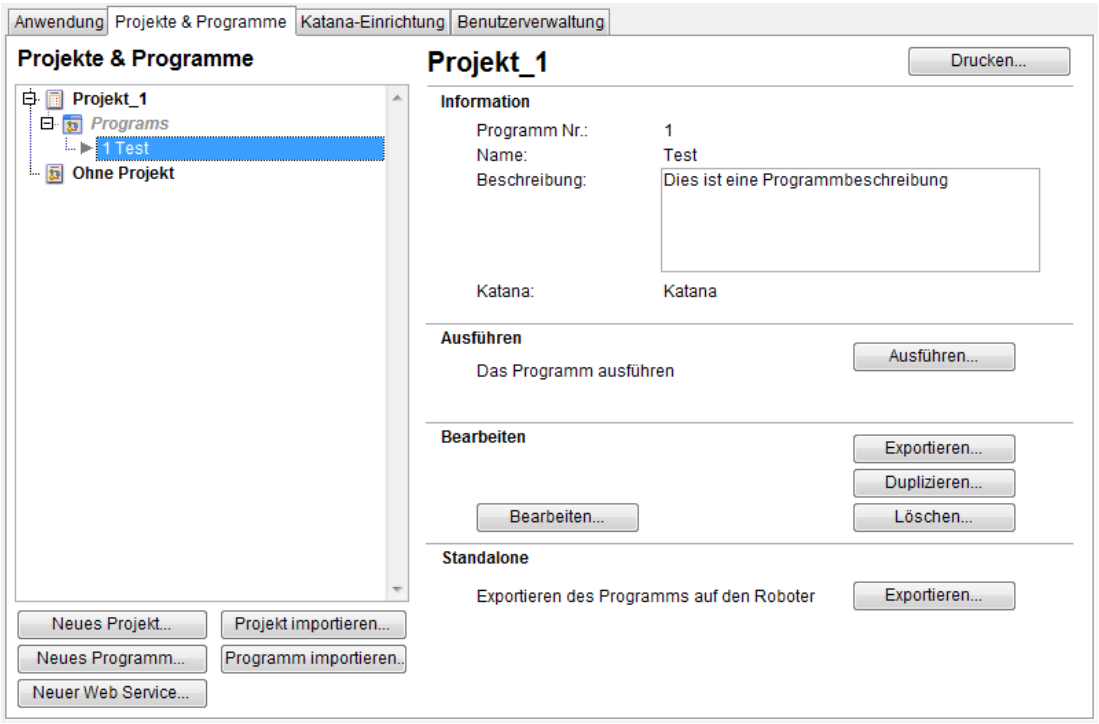
1.5.9 Suche

Mit Suchen kann in einem Programm ein bestimmter Codeausschnitt gesucht werden. Dabei werden alle Scripts des Programms durchsucht. In der untenstehenden Liste wird angegeben, in welchem Programm und in welchem Script des Programms sich der gesuchte Codeausschnitt befindet.



Falls die Checkbox *nur in dieses Programm durchsuchen* nicht aktiviert ist, wird in allen Programmen der Datenbank gesucht.

1.5.10 Programm-Funktionen



Folgende Funktionen stehen für ein Programm zur Verfügung:

- Ausführen...

  Das Programm wird ausgeführt. Die Schaltfläche Ausführen... verschwindet und Beenden... erscheint unterhalb. Nach der Beendigung des Programms erscheint Ausführen... wieder.

  Während ein Programm ausgeführt wird, kann es auch bearbeitet werden. Die gespeicherten Änderungen werden nach jedem abgeschlossenen Zyklus berücksichtigt. Das Projekt, auf das ein Programm zugreift, kann nicht geändert werden. Diese Daten können während der Ausführung nur gelesen werden.

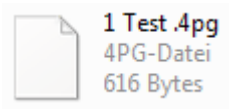
  Programme, die nicht auf den gerade verwendeten Katana zugreifen, können zusätzlich gestartet werden. Dazu muss das entsprechende Programm ausgewählt und auf Ausführen... geklickt werden.

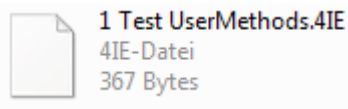
- Bearbeiten...

  Das Fenster ***Programm*** wird geöffnet.

- Exportieren...

  Das Programm wird mit allen dazugehörigen Scripts (Startscript, Endscript, Sequenzen und lokale Benutzermethoden) auf einem Datenträger gespeichert. Zuerst muss ein Speicherort angegeben werden. Danach wird ein Ordner mit dem Namen "*ProgrammNr ProgrammName*" (z.B. "*1 Test*") erstellt. Der Ordner beinhaltet die drei Scriptarten als Katana4D-Dateien (Start- und Endscript gehören zusammen).





- Duplizieren...

  Das Programm wird unter einem anderen Namen gespeichert. Eine neue Programm-Nummer wird vergeben.

- Löschen...

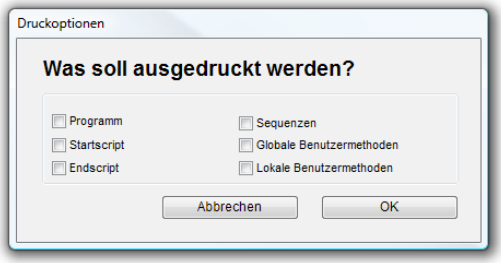
  Das Programm wird nach der Bestätigung einer Warnung aus der Datenbank gelöscht.

- Exportieren...

  Ein Stand-alone-Programm wird generiert. Der nächste Abschnitt enthält weitere Informationen.

- Drucken...

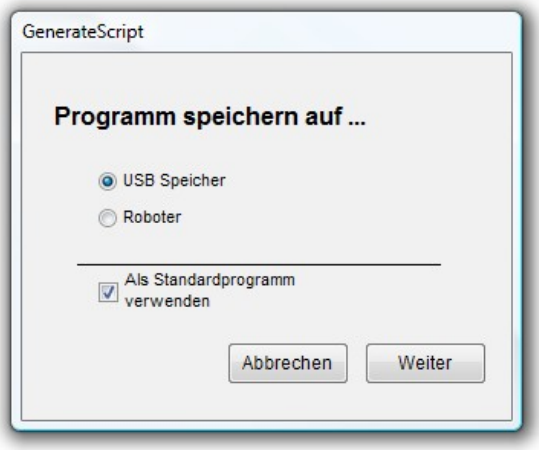
  Ein Dialog erscheint, in dem ausgewählt werden kann, was ausgedruckt werden soll. Nach dem Klicken auf OK wird die Auswahl ausgedruckt.



Die Druckoptionen beziehen sich nur auf das ausgewählte Programm.

1.5.11 Stand-alone

Ein Programm kann auf einen Katana oder einen USB-Speicher übertragen werden, indem das gewünschte Programm auswählt und danach die Schaltfläche Exportieren... gedrückt wird.



Das Programm kann direkt auf dem Roboter oder auf einem USB-Speicher gespeichert werden.

Falls das Programm standardmässig verwenden werden soll, muss dieses Auswahlfeld markiert werden.

Soll das Programm auf einen USB-Speicher geschrieben werden, öffnet sich mit Weiter eine neue Seite, auf der das Laufwerk ausgewählt werden kann. Ist der USB-Speicher noch nicht am Computer angeschlossen, sollte er jetzt verbunden werden. Das Programm wird automatisch gespeichert sobald der Speicher erkannt wurde.

Ausführen von Stand-alone Programmen

Stand-alone Programme können mit dem Controlpad ('PLAY') ausgeführt werden. Ausführlichere Informationen zu diesem Thema können dem Dokument “Katana450 Stand-alone Quickstart Guide“ entnommen werden.

Programmieren und Exportieren von Methoden als Webservice

Katana4D bietet die Möglichkeit, definierte Methoden als Webservice zu exportieren. Katana4D wird dann nur zum Programmieren benötigt eine externe Steuerung führt die Methoden über das Web direkt auf dem Katana aus.

Die Konfiguration und Programmierung von Webservices werden im Kapitel Schnittstellen genauer beschrieben.

Einschränkungen für den Stand-alone Betrieb

Es können nicht alle Katana4D Befehle im Stand-alone Betrieb verwendet werden. Insbesondere Befehle, die Berechnungen der Kinematik benötigen, können nicht verwendet werden. Grundsätzlich kann man aber jede Anwendung so erstellen, dass sie auch im Stand-alone Betrieb ausführbar ist.

In der Befehlsliste im Anhang ist ersichtlich, welche Befehle im Stand-alone Betrieb unterstützt werden.

Besonderheiten in der Programmierung für den Standalone-Betrieb

Für den Stand-alone Betrieb müssen abgesehen vom eingeschränkten Befehlsset einige Besonderheiten in der Programmierung berücksichtigt werden.

1. **Raster:**
   Raster benötigen für einen Export in den Stand-alone Modus eine besondere Behandlung. Die Rasterpositionen müssen beim Anfahren mit der Funktion 'MoveToGridOffsetPoint' oder 'MoveToGridOffsetPointLin' eingefasst werden, damit die Offsetpunkte generiert werden können:
   Beispiel:

```
VarInt1 := 1
Repeat
MoveToGridOffsetPointLin("r_offset1"; LinVelocity)
MoveToGridPointLin ("r1"; VarInt1; Velocity)
MoveToGridPointLinDelta ("r1"; VarInt1; 20; 20; 0; Velocity)
MoveToGridPointLin ("r1"; VarInt1; Velocity)
MoveToGridOffsetPointLin("r_offset1"; LinVelocity)
VarInt1 := VarInt1 + 1
Until(VarInt1 = GridSize+1)
```

2. **Multithreading:**
   Sollen mehrere Prozesse in ein Stand-alone Programm exportiert werden, können die einzelnen Programme mit dem Befehl 'NEW_PROCESS' als separater Thread definiert und exportiert werden.

Variablen

Variablen, die für eine lineare Bewegung verwendet werden, müssen direkt einen Wert enthalten. Verweise auf andere Variablen sind nicht zulässig.

Beispiel

```
Var1 := 100
Var2 := Var1
MoveToPointLin("p"; Var1)  \\funktioniert
MoveToPointLin("p"; Var2)  \\funktioniert nicht!
```

Lineare Bewegungen

Bei linearen Bewegungen im Stand-alone Betrieb ist zu berücksichtigen, dass diese nicht aus einer undefinierten Position gestartet werden können. Bringt man den Roboter mit dem Befehl 'MoveAxis' in eine beliebige Position, kann von dieser aus keine lineare Bewegung gestartet werden. Das Gleiche gilt auch, wenn man vor der linearen Bewegung einen Weg ohne Endpunkt abfährt.

Generell muss vor einer Linear-Bewegung ein definierter Punkt (Punkte aus dem Projekt) angefahren werden.

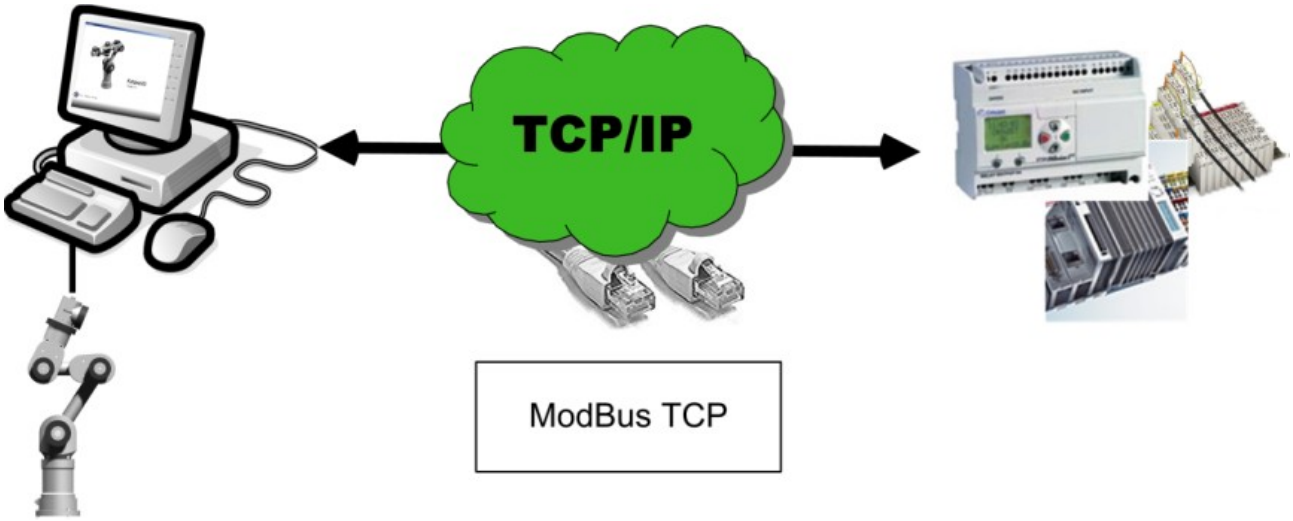
1.6 Schnittstellen

1.6.1 ModbusTCP

Kurzbeschreibung

ModbusTCP ist eine feldbusbasierte I/O-Kommunikation über das standardisierte ModbusTCP Protokoll. Dies ist die von Neuronics empfohlene Schnittstelle zur Erweiterung der Peripherie sowie zur Feldbusanbindung.

Schema



Vor- und Nachteile

+ sehr flexibler Einsatz möglich
+ schnelle Datenübertragung (<5ms pro K4D-Befehl möglich, abhängig vom Endgerät)
+ ermöglicht Datenaustausch zwischen Katana und I/O-Modulen, SPS etc. verschiedener Hersteller (sofern ModbusTCP unterstützt wird).

Anforderungen

- freier Netzwerkanschluss
- ModbusTCP Server (ModbusTCP-fähiges Endgerät).

Technische Beschreibung

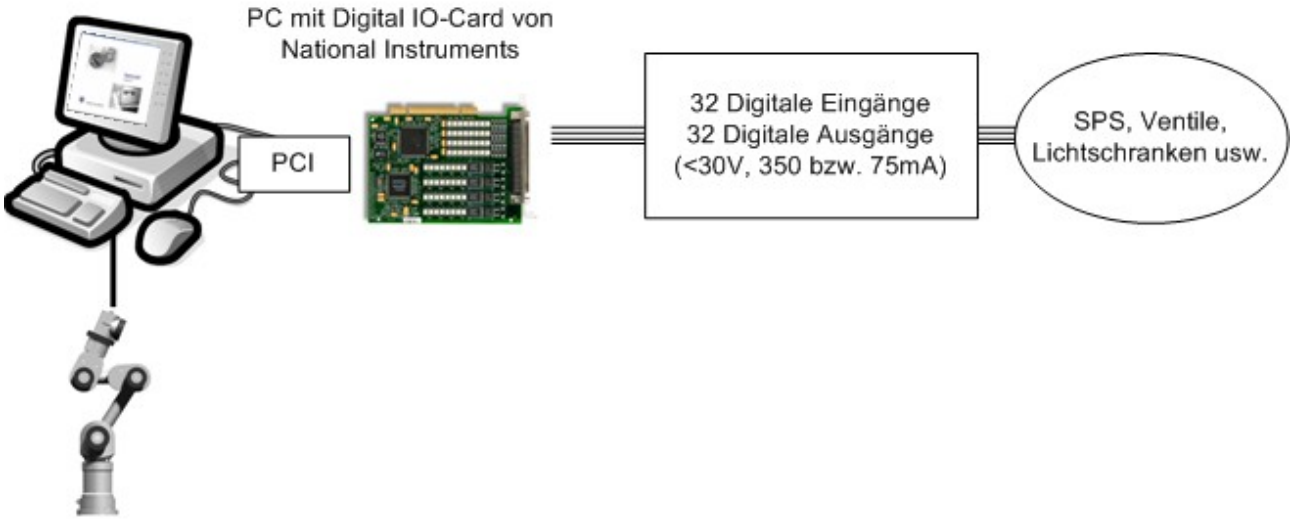
Aus Katana4D können Zahlen (pro Adresse ein Wort, 2 Bytes) oder einzelne Bits geschrieben und gelesen werden.

1.6.2 National Instruments Digital I/O Card

Kurzbeschreibung

Kommunikation über digitale Ein- oder Ausgänge einer I/O-Karte von National Instruments.

Schema



Vor- und Nachteile

+ viele digitale Ein- und Ausgänge
+ kurze Lese- und Schreibzeiten
+ einfache Ansteuerung von nicht intelligenten Systemen (Ventile, Lichtschranken, etc.)
- zusätzliche Karte oder Box (je nach Typ) nötig, die am PC angeschlossen sein muss.

Anforderungen

- NI 6514 (PCI), NI DAQPad-6015 (USB) oder NI USB-6501.

Technische Beschreibung

Die gängigste Lösung basiert auf einer PCI-Karte mit 32 digitalen Ein- und Ausgängen, die extern mit maximal 30V (75 mA bis 350 mA pro Ausgang) gespeist werden können. Mit der Karte NI 6514 sind Zugriffszeiten von 8 ms möglich.

Falls ein Laptop für die Steuerung des Katanas zuständig ist, kann mit dem NI DAQPad-6015 (8 digitale Ein- und Ausgänge, 16 analoge Eingänge, 2 analoge Ausgänge) oder NI DAQPad-6016 (32 digitale Ein- und Ausgänge, 16 analoge Eingänge, 2 analoge Ausgänge) gearbeitet werden. Dies ist eine externe Box, die über USB mit dem Notebook verbunden ist.

Beschreibung der In- und Outputs

NI 6514 (PCI)

Input K4D	Pin		GND
1	1	P0.0	9,10,11,12
2	2	P0.1	
3	3	P0.2	
4	4	P0.3	
5	5	P0.4	
6	6	P0.5	
7	7	P0.6	
8	8	P0.7	
9	13	P2.0	21,22,23,24
10	14	P2.1	
11	15	P2.2	
12	16	P2.3	
13	17	P2.4	
14	18	P2.5	
15	19	P2.6	
16	20	P2.7	
17	51	P1.0	59,60,61,62
18	52	P1.1	
19	53	P1.2	
20	54	P1.3	
21	55	P1.4	
22	56	P1.5	
23	57	P1.6	
24	58	P1.7	
25	63	P3.0	71,72,73,74
26	64	P3.1	
27	65	P3.2	
28	66	P3.3	
29	67	P3.4	
30	68	P3.5	
31	69	P3.6	
32	70	P3.7	

Output K4D	Pin		GND	VCC
1	26	P4.0	34	35,36,37
2	27	P4.1		
3	28	P4.2		
4	29	P4.3		
5	30	P4.4		
6	31	P4.5		
7	32	P4.6		
8	33	P4.7		
9	38	P6.0	46	47,48,49
10	39	P6.1		
11	40	P6.2		
12	41	P6.3		
13	42	P6.4		
14	43	P6.5		
15	44	P6.6		
16	45	P6.7		
17	76	P5.0	84	85,86,87
18	77	P5.1		
19	78	P5.2		
20	79	P5.3		
21	80	P5.4		
22	81	P5.5		
23	82	P5.6		
24	83	P5.7		
25	88	P7.0	96	97,98,99
26	89	P7.1		
27	90	P7.2		
28	91	P7.3		
29	92	P7.4		
30	93	P7.5		
31	94	P7.6		
32	95	P7.7		

NI DAQPad-6015 (USB)

Input K4D	Pin		GND
1	33	P0.0	35
2	34	P0.1	
3	36	P0.2	
			39
4	37	P0.3	
5	38	P0.4	
6	40	P0.5	

Output K4D	Pin		GND
1	33	P0.0	35
2	34	P0.1	
			39
3	36	P0.2	
4	37	P0.3	
5	38	P0.4	
6	40	P0.5	

7	41	P0.6	43
8	42	P0.7	

7	41	P0.6	43
8	42	P0.7	

NI USB-6501

Input K4D	Pin		GND
1	17	P0.0	25,26
2	18	P0.1	
3	19	P0.2	
4	20	P0.3	
5	21	P0.4	
6	22	P0.5	
7	23	P0.6	
8	24	P0.7	
9	27	P1.0	32
10	28	P1.1	
11	29	P1.2	
12	30	P1.3	
13	6	P1.4	1
14	5	P1.5	
15	4	P1.6	
16	3	P1.7	
17	16	P2.0	7,8
18	15	P2.1	
19	14	P2.2	
20	13	P2.3	
21	12	P2.4	
22	11	P2.5	
23	10	P2.6	
24	9	P2.7	

Output K4D	Pin		GND
1	17	P0.0	25,26
2	18	P0.1	
3	19	P0.2	
4	20	P0.3	
5	21	P0.4	
6	22	P0.5	
7	23	P0.6	
8	24	P0.7	
9	27	P1.0	32
10	28	P1.1	
11	29	P1.2	
12	30	P1.3	
13	6	P1.4	1
14	5	P1.5	
15	4	P1.6	
16	3	P1.7	
17	16	P2.0	7,8
18	15	P2.1	
19	14	P2.2	
20	13	P2.3	
21	12	P2.4	
22	11	P2.5	
23	10	P2.6	
24	9	P2.7	

Steuerung in Katana4D

Wenn eine NI-Karte erkannt wurde, erscheint ein neues Register im Hauptformular.

In der linken Spalte sind alle erkannten Karten aufgelistet. Es wird bei jeder erkannten Karte die Geräte-Nummer sowie der Typ angezeigt. Die Karte kann durch einen Klick auf den entsprechenden Eintrag ausgewählt werden.

Oben können die Eingänge gelesen werden. Durch Drücken der Schaltfläche Jetzt lesen werden die Eingänge aktualisiert. Darunter können die Ausgänge gesetzt werden. Die Befehle, um im Programmablauf auf die NI-Karte(n) zuzugreifen, sind im Kapitel Befehlsbeschreibung erläutert.

Spezielle Hinweise

Auf Anfrage können auch andere Karten bzw. Geräte von National Instruments unterstützt werden. Bis jetzt werden aber nur die oben genannten Karten eingesetzt.

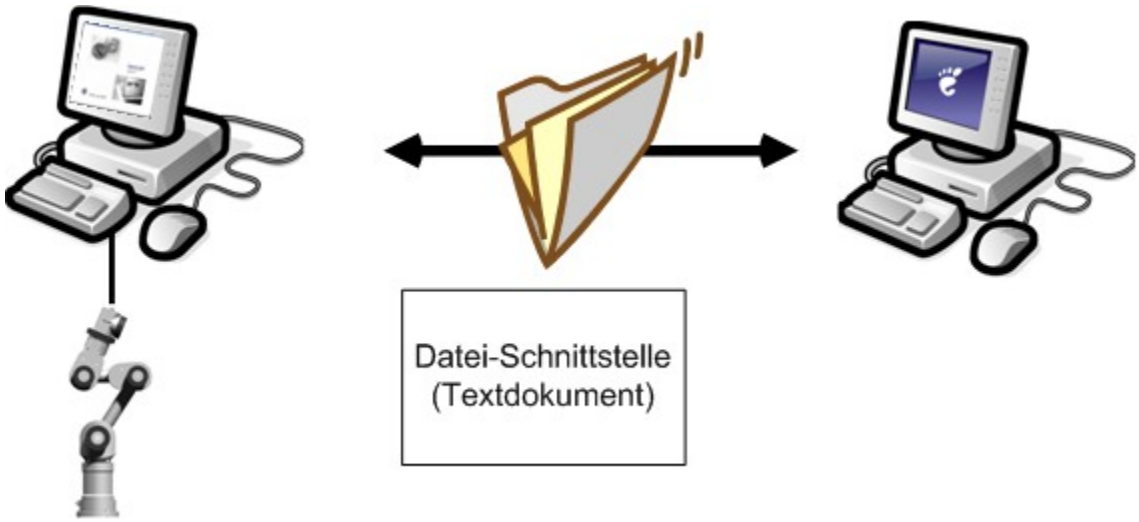
Beim NI DAQPad-6015 (USB) können bis jetzt nur die digitalen Ein- und Ausgänge angesprochen werden, aber nicht die analogen.

1.6.3 Datei Kommunikation

Kurzbeschreibung

Kommunikation über eine Textdatei.

Schema



Vor- und Nachteile

+ einfache Programmierung
+ sehr flexibel
+ keine Hardware-Anforderungen
+ netzwerkfähig
- externes System muss Datei-Handling unterstützen
- Zugriff muss geregelt werden (Schreibe- und Lesezugriff).

Anforderungen

- externes intelligentes System, das Datei-Handling unterstützt.

Technische Beschreibung

Verschiedene Rechner oder Prozesse greifen auf eine gemeinsame Textdatei zu. In der Textdatei können beliebige Zeichen gelesen oder geschrieben werden.

Spezielle Hinweise

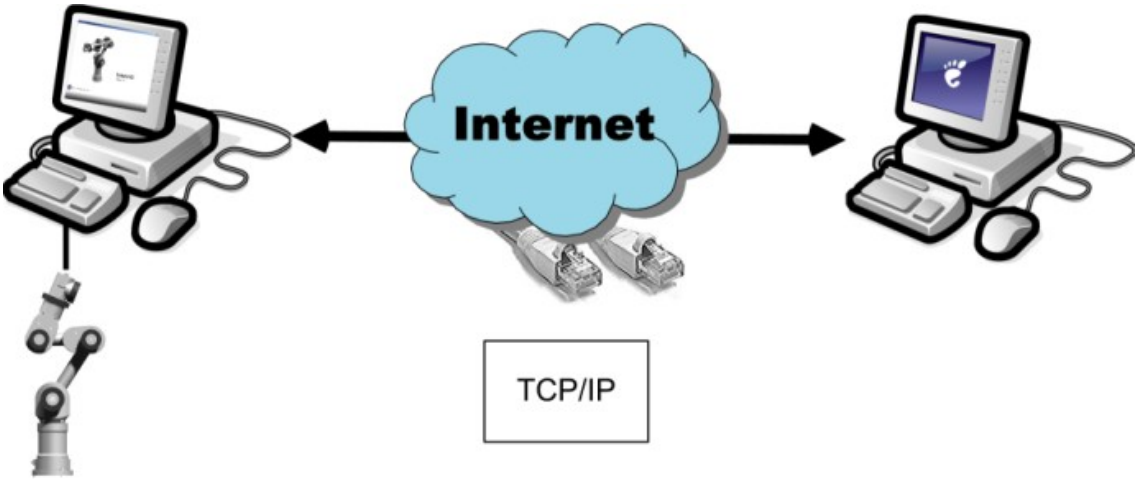
Die Datei-Schnittstelle kann z.B. eingesetzt werden, um mit einer Kamera zu kommunizieren.

1.6.4 TCP/IP

Kurzbeschreibung

Kommunikation über das Netzwerkprotokoll TCP/IP.

Schema



Vor- und Nachteile

+ sehr flexibler Einsatz möglich
+ schnelle Datenübertragung
+ mehrere externe Geräte können angesteuert werden.

Anforderungen

- freier Netzwerkanschluss
- externes System, das die Kommunikation über TCP/IP unterstützt.

Technische Beschreibung

Aus Katana4D können Zeichen empfangen und gesendet werden.

Spezielle Hinweise

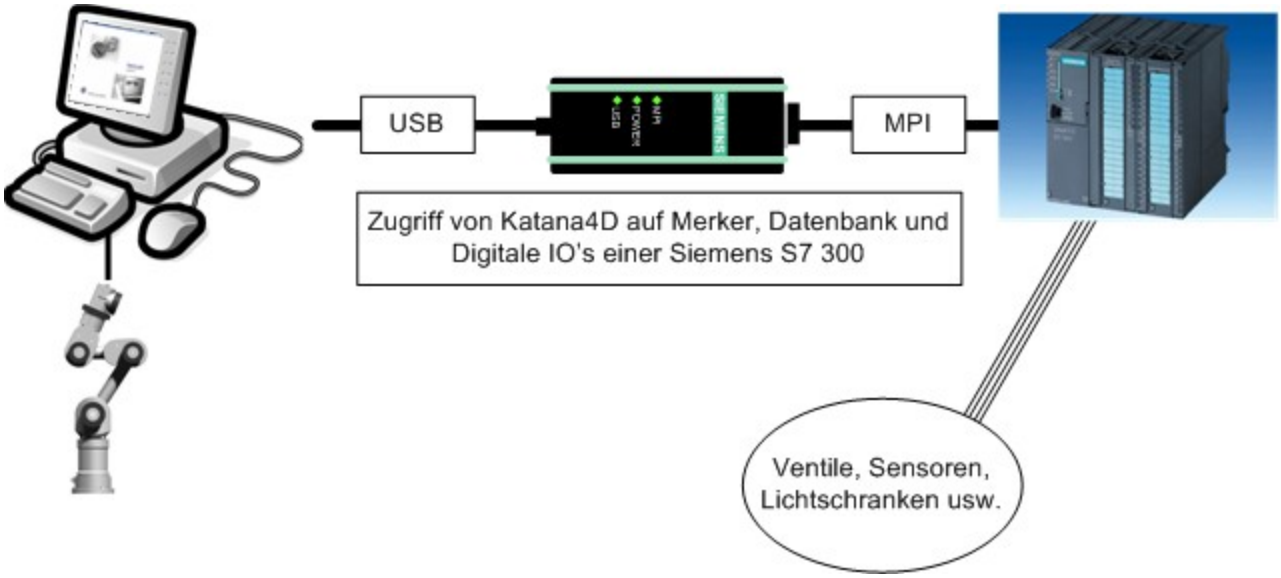
Die TCP/IP-Schnittstelle kann z.B. eingesetzt werden, um mit einer Kamera zu kommunizieren.

1.6.5 Siemens S7 – Plugin

Kurzbeschreibung

Kommunikation mit einer Siemens S7 3xx oder 4xx.

Schema


USB
MPI
Zugriff von Katana4D auf Merker, Datenbank und Digitale IO's einer Siemens S7 300
Ventile, Sensoren, Lichtschranken usw.

Vor- und Nachteile

+ einfacher Informationsaustausch zwischen Katana4D und einer Siemens S7 3xx oder 4xx
+ es kann auf jegliche Daten der Siemens S7 3xx oder 4xx zugegriffen werden
- hohe Lese- und Schreibzeit (ca. 45 ms pro Befehl)
- zusätzliche Box nötig, die immer am PC angeschlossen sein muss.

Anforderungen

- Siemens S7 3xx oder 4xx
- Siemens S7 PC Adapter USB (Produkt-Nr: 6ES7972-0CB20-0XA0).

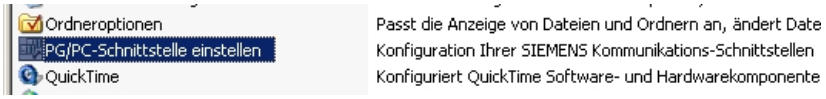
Technische Beschreibung

Die Daten der SPS werden über einen Standard-Programmieradapter ausgetauscht. Am Computer ist der Adapter an einem USB-Port angeschlossen und an der Siemens S7 3xx / 4xx an der MPI-Schnittstelle. Katana4D kann über vordefinierte Befehle auf alle Daten (Merker, digitale Ein/Ausgänge, Datenbank) der SPS zugreifen.

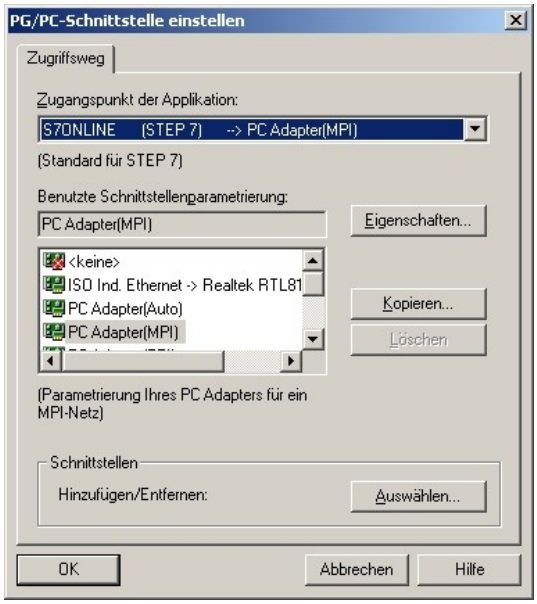
Spezielle Hinweise

Für den sicheren Betrieb des USB-Adapters muss die Treiberversion 1.2 installiert sein.

In der Systemsteuerung muss die „PG/PC-Schnittstelle“ unter Umständen noch angepasst werden.



Als “Zugangspunkt der Applikation“ wird der Eintrag “S7ONLINE“ ausgewählt. Die “Benutzte Schnittstellenparametrierung“ muss “PC Adapter (MPI)“ sein.



Unter den Eigenschaften vom PC Adapter(MPI) muss als lokaler Anschluss “USB“ ausgewählt werden.

1.6.6 Web-Services

Kurzbeschreibung

Exportieren von Programmen als Webservice und direkte, webbasierte Kommunikation mit dem Katana über ein Netzwerk (Internet/Intranet).

Schema

Vor- und Nachteile

+ Methoden und Programme werden direkt über das Netzwerk aufgerufen
+ Aufrufe der Methoden werden vom externen System übernommen
- langsame Kommunikation
- nicht der ganze Befehlssatz verfügbar.

Anforderungen

- externes intelligentes System das Web-Services unterstützt
- Netzwerkverbindung.

Technische Beschreibung

Katana4D ermöglicht das Definieren von Methoden, die dann als Webservice auf den Katana exportiert werden können. Diese Methoden können im Stand-alone Modus direkt auf dem Katana aufgerufen werden. Katana4D wird dann nur noch zur Programmierung, nicht mehr als Steuerung benötigt. Es werden die Protokolle SOAP und XML-RPC unterstützt.

2 Befehlsbeschreibung

Befehlsverzeichnis

Aktionsbefehle.....79

Allgemeine Aktionsbefehle.....80

MoveAxis(AxisNr; Angle) *.....80
WaitForAxis (AxisNr; Angle; Tolerance) *.....80
MoveAxisAndWait (AxisNr; Angle; Tolerance) *.....81
AxisOff (AxisNr) *.....81
AxisFreeze (AxisNr) *.....81
Calibrate *.....82
AllAxesOff *.....82
FreezeAllAxes *.....82
OpenGripper {(Velocity)} *.....82
CloseGripper {(Velocity)} *.....82
OpenGripperDelta (DeltaAngle{; Velocity}) *.....83
CloseGripperDelta (DeltaAngle{; Velocity}) *.....83

Anfahren von Positionen im Raum.....84

MoveToPos (X; Y; Z; Phi; Theta; Psi {; Velocity; Acceleration}).....84
MoveToPosLin (X; Y; Z; Phi; Theta; Psi {; LinVelocity}).....84
MoveToPosDeg(Angle1; Angle2; Angle3; Angle4; Angle5; Angle6{; Velocity; Acceleration})84
MoveDelta (DeltaX; DeltaY; DeltaZ{; Velocity; Acceleration}).....85
MoveDeltaLin (DeltaX; DeltaY; DeltaZ{; LinVelocity}).....85
MoveRadial (Distance {; Velocity; Acceleration}).....86
MoveRadialLin (Distance {; LinVelocity}).....86
MoveInToolDirection (DeltaX; DeltaY; DeltaZ ; Velocity; Acceleration).....86
MoveInToolDirectionLin (DeltaX; DeltaY; DeltaZ ; LinVelocity).....88

Anfahren von Punkten im Raster.....89

MoveToGridPoint (Name; PointNr; Velocity; Acceleration) *.....89
MoveToGridPointDelta (“Name“; PointNr; DeltaX; DeltaY; DeltaZ; Velocity; Acceleration) *. 90
MoveToGridPointLin (“Name“; PointNr; LinVelocity) *.....90
MoveToGridPointLinDelta (“Name“; PointNr; DeltaX; DeltaY; DeltaZ; LinVelocity) *.....90

Anfahren von Punkten im Raum.....91

MoveToPoint (“Name“; Velocity; Acceleration) *.....91
MoveToPointDelta (“Name“; DeltaX; DeltaY; DeltaZ; Velocity; Acceleration).....92
MoveToPointDeg (“Name“; Velocity; Acceleration) *.....92
MoveToPointLin (“Name“; LinVelocity) *.....93
MoveToPointLinDelta (“Name“; DeltaX; DeltaY; DeltaZ; LinVelocity).....93
MoveToPointArc (“StartPoint“; “EndPoint“; “HelpPoint“; LinVelocity) *.....94

Abfahren der Wege.....95

MoveAlongPath (“Name“{; PathVelocity{; Direction}}) *.....95

Abfahren der Trajektorien.....96

MoveTrajectory (“Name“{; Velocity{; Direction}}) *.....96

Lesen und Speichern.....97

Punkte.....97
GetPoint (“Name“; ->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi).....97

GetPointDeg (“Name“; ->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6).....97
SavePoint (“Name“; X; Y; Z; Phi; Theta; Psi) -> Boolean.....98
SavePointDeg (“Name“; Ang1; Ang2; Ang3; Ang4; Ang5; Ang6) -> Boolean.....99
Raster.....99
GetGridPoint (“Name“; PointNr; ->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi).....99
GetGridPointDeg (“Name“; PointNr; ->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6) 100
SaveGridPoint (“Name“; PointNr; X; Y; Z; Phi; Theta; Psi) -> Boolean.....101
SaveGridPointDeg(“Name“; PointNr; Ang1; Ang2; Ang3; Ang4; Ang5; Ang6)->Boolean 102
Benutzervariablen.....102
GetIntVar (“Name“) -> Integer.....102
GetRealVar (“Name“) -> Real.....103
GetStrVar (“Name“) -> String.....103
SetIntVar (“Name“; Value).....103
SetRealVar (“Name“; Value).....104
SetStrVar (“Name“; Value).....104

Zustandsabfragen.....105

ReadCoordinates (->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi).....105
ReadAxisPos (AxisNr) * -> Integer.....105
ReadPositions (->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6).....106
ReadAxisState (AxisNr) * -> Integer.....106
ReadMaxVel (AxisNr{; ->VarInt2; ->VarInt3}) -> Integer.....107
ReadMaxPWM (AxisNr{; ->VarInt2; ->VarInt3}) -> Integer.....107
ReadMaxAcc (AxisNr) -> Integer.....108
ReadCollisionPosLimit (AxisNr) -> Integer.....108
ReadCollisionPosLimitLin (AxisNr) -> Integer.....108
ReadK (Motornummer; ->VarInt1; ->VarInt2; ->VarInt3).....109
ReadSensor(SensorNr) -> Integer.....109

Schnittstellen.....110

National Instruments IO Card.....110
NI_ReadInput (InputNr) -> Integer.....110
NI_SetOutput (OutputNr; Value).....110
NI_ReadBank (BankNr) -> Integer.....111
NI_SetBank(BankNr; Value).....111

Datei-Kommunikation.....112
WriteFile (“FileName“; Value) -> Boolean.....112
AddToFile (FileName; Value{; CreateDocument}) -> Boolean.....112
ReadFile (“FileName“; DocumentPosition; NumChar{; StopChar}) -> String.....113

Serielle Kommunikation.....114
SetComport (ComNr).....114
SendSerial (VarStr1).....114
ReceiveSerial -> Text.....115

TCP/IP.....116
TCP_IP_Connect (“Hostname“; Portnumber{; Timeout}) * -> Integer.....116
TCP_IP_OpenServer (Portnumber{; Timeout}) * -> Integer.....116
TCP_IP_CloseConnection (TCPSessionID) *.....117
TCP_IP_Ping (Hostname; Timeout) -> Integer.....117
TCP_IP_Receive (TCPSessionID) * -> Text.....117

TCP_IP_Send (TCPSessionID; SendText) *....117

ModBusTCP....119

ModBusTCP_Connect (Host-IP{; Timeout}{; Port}{; OutputOffset}) *....119
ModBusTCP_Disconnect *....119
ModBusTCP_ReadWord (Address) * -> Integer....120
ModBusTCP_WriteWord (Address; Value) *....120
ModBusTCP_ReadBit (Address; Position) * -> Integer....121
ModBusTCP_WriteBit (Address; Position; Status) *....121
Beckhoff_Watchdog_Enable {(Timeout)} *....121
Beckhoff_Watchdog_Disable *....122

Siemens S7 - Plugin....123

S7MPI_Connect (MPI-Address)....123
S7MPI_Disconnect....123
S7MPI_DBReadByte (DBNr; DBBNr) -> Byte....123
S7MPI_DBReadWord (DBNr; DBWNr) -> Integer....124
S7MPI_DBReadDWord (DBNr; DBWNr) -> 32bit Integer....124
S7MPI_DBReadString (DBNr; DBBNr) -> Char....125
S7MPI_DBWriteByte (DBNr; DBBNr; Value)....125
S7MPI_DBWriteWord (DBNr; DBWNr; Value)....126
S7MPI_DBWriteDWord (DBNr; DBDNr; Value)....126
S7MPI_DBWriteString (DBNr; DBBNr; Value)....127
S7MPI_ReadDigitalInputByte (ByteNr) -> Integer....127
S7MPI_ReadDigitalOutputByte (ByteNr) -> Integer....128
S7MPI_ReadDigitalInputBit (ByteNr; BitNr) -> Boolean....128
S7MPI_ReadDigitalOutputBit (ByteNr; BitNr) -> Boolean....129
S7MPI_WriteDigitalOutputByte (ByteNr; Value)....129
S7MPI_ReadFlagBit (ByteNr; BitNr) -> Boolean....130
S7MPI_ReadFlagByte(ByteNr) -> Integer....130
S7MPI_ReadFlagWord (ByteNr) -> Integer....131
S7MPI_ReadFlagDWord (ByteNr) -> Integer....131
S7MPI_ReadFlagString (ByteNr) -> Char....131
S7MPI_WriteFlagBit (ByteNr; BitNr; Value)....132
S7MPI_WriteFlagByte (ByteNr; Value)....132
S7MPI_WriteFlagWord (ByteNr; Value)....132
S7MPI_WriteFlagDWord (ByteNr; Value)....133
S7MPI_WriteFlagString(ByteNr; Value)....133
CB_SetDigitalOutput (OutputNr; Value) *....134
CB_ReadDigitalInput (InputNr) * -> Integer....134
CB_SetLED (“State“) *....135

Programmablauf-Steuerung....136

Break *....136
If Then Else *....136
Repeat Until *....137
Wait (Millisekunden) *....137
RunUserMethod (“UserMethodName“) *....137
GetErrorMessage -> String....138
Login ({User{; Password}})....138
Logout....138
Alert (“Text“)....139
Confirm (“Text“{; “OKTitle“{; “CancelTitle“}})....139
Request (“Message“{; “DefaultResponse“{; “OKTitle“{; “CancelTitle“}}}) -> String....140

PostMessage (“Message“).....140
OpenWindow (Width; Height; “WindowTitle“).....140
RunProgram (Nr; Repetitions).....141
RunApplication (“Name“).....141
ExitProgram (Nr).....141
ExitApplication (“Name“).....141
PauseProgram (Nr{; Timeout}) -> Integer.....142
ResumeProgram (Nr) -> Integer.....142
ProgramState (Nr) -> Integer.....143

Mathematische Funktionen.....144

Abs (Value) * -> Numerisch.....144
Arctan (Angle) -> Numerisch.....144
Cos (Angle) -> Numerisch.....144
Dec (Value) -> Numerisch.....145
Exp (Value) -> Numerisch.....145
Int (Value) * ->Numerisch.....145
Log (Value) -> Numerisch.....145
Mod (Value; Divisor) -> Numerisch.....146
Random -> Numerisch.....146
Round (Value; Places) * -> Numerisch.....146
Sin (Angle) -> Numerisch.....147
Square root (Value) -> Numerisch.....147
Tan (Angle) -> Numerisch.....147
Trunc (Value; Places) -> Numerisch.....148

Datum & Zeit.....149

Add to date (Date; Year; Month; Day) -> Datum.....149
Current date -> Datum.....149
Current time -> Zeit.....149
Date (Text) -> Datum.....150
Day of (Date) -> Numerisch.....150
Day number (Date) -> Numerisch.....150
Milliseconds -> LongInteger.....150
Month of (Date) -> Numerisch.....151
Time (Text) -> Zeit.....151
Time string (Time) -> String.....151
Year of (Date) ->Numerisch.....151

String.....152

Ascii (Character) -> Numerisch.....152
Change string (Source; Destination; Position) -> String.....152
Char (Nr) -> String.....152
Delete string (Source; Position; Length) -> String.....153
Insert string (Source; Destination; Position) * -> String.....153
Length * (Text) -> Numerisch.....153
Lowercase (Text) -> String.....154
Num (Expression) * -> Numerisch.....154
Position (Find; String) -> Numerisch.....154
Replace string (Source; Old; New{; Size}) -> String).....155
String (Expression{; Format}) * -> String).....155
Substring (Source; FirstChar{; Length}) -> String.....155

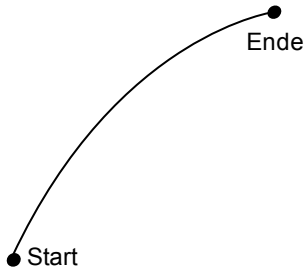
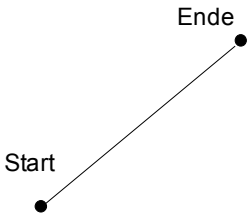
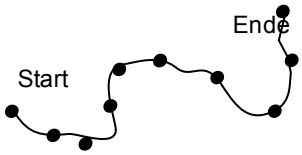
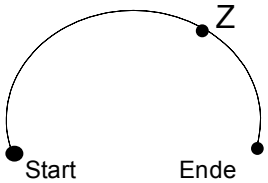
Uppercase (Text) -> String....156

Array....157

ARRAY BOOLEAN (ArrayName; Size{; Size2})....157
ARRAY DATE (ArrayName; Size{; Size2})....157
ARRAY INTEGER (ArrayName; Size{; Size2})....158
ARRAY LONGINT (ArrayName; Size{; Size2})....158
ARRAY REAL (ArrayName; Size{; Size2})....158
ARRAY STRING (Length; ArrayName; Size{; Size2})....159
ARRAY TEXT (ArrayName; Size{; Size2})....159
Size of array (Array) -> Numerisch....159
SORT ARRAY (Array{; Array2; ...; ArrayN}{; > or <})....160
Find in array (Array; Value{; Start}) -> Numerisch....160
INSERT ELEMENT (ArrayName; Position{; HowMany}) 160
DELETE ELEMENT (ArrayName; Position{; HowMany}) 161
COPY ARRAY (Source; Destination)....161
Count in array (Array; Value) -> LongInteger....161
APPEND TO ARRAY (Array; Value)....162

2.1 Aktionsbefehle

Überblick

Funktionen	Grafik	Greiferorientierung	Bemerkungen
Nicht lineare Bewegung MoveToPos MoveToPosDeg MoveDelta MoveRadial MoveInToolDirection MoveToGridPoint MoveToGridPointDelta MoveToPoint MoveToPointDelta		Die Orientierung des Greifers ist im Start- und Endpunkt bestimmt. Dazwischen ist sie jedoch unbestimmt.	Die Bewegung wir d so berechnet, dass alle Gelenke sich etwa gleich lange bewegen. Wenn nahe einer Geraden oder Fläche gefahren werden soll, sollte eine lineare Bewegung verwendet werden.
Lineare Bewegung MoveToPosLin MoveDeltaLin MoveRadialLin MoveInToolDirectionLin MoveToGridPointLin MoveToGridPoinLinDelta MoveToPointLin MoveToPointLinDelta		Die Greiferorientierung ist in Start- und Endpunkt bestimmt. Dazwischen ändert sich die Orientierung linear.	Die Linearbahn bezieht sich auf den Tool Center Point. Sollte es aufgrund unerreichbarer Zwischenpunkte nicht möglich sein, die Bahn zu fahren, erscheint eine Fehlermeldung. In diesem Fall kann der Punkt stattdessen mit einer nicht linearen Bewegung angefahren werden.
Wege MoveAlongPath		Die Orientierung des Greifers ist bei der Wegfunktion davon abhängig wie der Weg aufgenommen wurde.	Der Roboter merkt sich während der Aufzeichnung des Weges sehr viele Punkte inklusive der Greiferorientierung.
Kreisbewegung MoveToPointArc		Die Greiferorientierung bei der Kreisbewegung wird durch die drei Punkte berechnet. Der Roboter ändert die Orientierung proportional zum zurückgelegten Weg und ist am Schluss so orientiert wie der Einzelpunkt, den er gerade anfährt.	Sie geben einen Start- und Endpunkt sowie einen Zwischenpunkt (Z) an. Der Roboter fährt in einer Kreisbewegung vom Start- zum Endpunkt.

Start

2.1.1 Allgemeine Aktionsbefehle

MoveAxis

MoveAxis(AxisNr; Angle) *

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 - 6)
Angle	Real	Zielposition in Grad (siehe Katana 450 Manual)

Beschreibung
'MoveAxis' ist der einfachste und schnellste Achsenbefehl. Die entsprechende Achse fährt zur angegebenen Zielposition. Sofort, nachdem der Befehl abgesetzt wurde, kann das Programm fortgesetzt werden. Abweichende Anweisungen an der gleichen Achse werden sofort ausgeführt, unabhängig davon, ob die Achse das vorherige Ziel bereits erreicht hat oder nicht.

Beispiel
MoveAxis (2; 30)

Bewegt Motor 2 in die 30° Position

WaitForAxis

WaitForAxis (AxisNr; Angle; Tolerance) *

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 - 6)
Angle	Real	Zielposition in Grad (siehe Katana 450 Manual)
Tolerance	Integer	Toleranz in Grad (Standard: 1 Grad)

Beschreibung
'WaitForAxis' wartet bis die Achse die Zielposition abzüglich Toleranz erreicht hat. Erst dann ist der Befehl abgearbeitet und der nächste Befehl im Script kann ausgeführt werden.

Beispiel
MoveAxis (1; 30)
WaitForAxis (1; 30; 1)

Mit den Befehlen 'MoveAxis' und 'WaitForAxis' wird der Befehl 'MoveAxisAndWait' realisiert.

MoveAxisAndWait

MoveAxisAndWait (AxisNr; Angle; Tolerance) *

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 - 6)
Angle	Real	Zielposition in Grad (siehe Katana 450 Manual)
Tolerance	Integer	Toleranz in Grad (Standard: 1 Grad)

Beschreibung
'MoveAxisAndWait' erlaubt eine Bewegung auszuführen und die nächste Aktion solange zu verzögern, bis die Toleranz zur Zielposition unterschritten wurde.

Beispiel
MoveAxisAndWait (2; 30; 1)

Achse 2 wird in die 30° Position gefahren. Es wird gewartet, bis die Achse die Zielposition abzüglich der angegebenen Toleranz erreicht hat.

AxisOff

AxisOff (AxisNr) *

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 - 6)

Beschreibung
Schaltet die angegebene Achse aus. Diese Funktion kann nützlich sein, wenn beim Ergreifen eines Objektes der Katana in bestimmten Achsen nachgeben soll. Die Achse wird durch einen anderen Aktionsbefehl wieder eingeschaltet (z.B. MoveAxis oder AxisFreeze).

AxisFreeze

AxisFreeze (AxisNr) *

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 - 6)

Beschreibung
'AxisFreeze' fixiert die entsprechende Achse an der aktuellen Position. Diese Funktion ist nützlich, um die Achse auf die aktuelle Position zu setzen, sodass sie dort verharrt. Durch den Freeze-Befehl wird die aktuelle Position zur Zielposition.

Calibrate

Calibrate *

Beschreibung
'Calibrate' führt eine vollständige Hauptjustierung durch. Sie erfolgt gleich wie die durch den Button “Hauptjustierung“ im Katana-Dialog, ausser dass keine Meldung am Schluss der Justierung ausgegeben wird. Achtung: Der Roboter muss zuvor in eine sichere Lage per Hand oder per Programmierung gebracht werden.

AllAxesOff

AllAxesOff *

Beschreibung
Dieser Befehl schaltet alle Achsen aus. Um die Achsen wieder zu fixieren, kann der Befehl 'FreezeAllAxes' verwendet werden.

FreezeAllAxes

FreezeAllAxes *

Beschreibung
'FreezeAllAxes' fixiert alle Achsen an der aktuellen Position. Um die Achsen zu lösen, kann der Befehl 'AllAxesOff' verwendet werden.

OpenGripper

OpenGripper {(Velocity)} *

Parameter	Typ	Beschreibung
Velocity	Integer	Geschwindigkeit in Prozent

Beschreibung
'OpenGripper' öffnet den Greifer so weit wie möglich. Über den optionalen Parameter 'Velocity' kann die Geschwindigkeit der Bewegung angepasst werden.

CloseGripper

CloseGripper {(Velocity)} *

Parameter	Typ	Beschreibung
Velocity	Integer	Geschwindigkeit in Prozent

Beschreibung
'CloseGripper' schliesst den Greifer so weit wie möglich. Über den optionalen Parameter 'Velocity' kann die Geschwindigkeit der Bewegung angepasst werden.

OpenGripperDelta

OpenGripperDelta (DeltaAngle{; Velocity}) *

Parameter	Typ	Beschreibung
DeltaAngle	Real	Winkel in Grad
Velocity	Integer	Geschwindigkeit in Prozent

Beschreibung
'OpenGripperDelta' öffnet den Greifer um den Winkel 'DeltaAngle'. Über den optionalen Parameter 'Velocity' kann die Geschwindigkeit der Bewegung angepasst werden.

CloseGripperDelta

CloseGripperDelta (DeltaAngle{; Velocity}) *

Parameter	Typ	Beschreibung
DeltaAngle	Real	Winkel in Grad
Velocity	Integer	Geschwindigkeit in Prozent

Beschreibung
'CloseGripperDelta' schliesst den Greifer um den angegebenen Winkel 'DeltaAngle'. Über den optionalen Parameter 'Velocity' kann die Geschwindigkeit der Bewegung angepasst werden.

2.1.2 Anfahren von Positionen im Raum

MoveToPos

MoveToPos (X; Y; Z; Phi; Theta; Psi {; Velocity; Acceleration})

Parameter	Typ	Beschreibung
X	Real	Raumkoordinate X
Y	Real	Raumkoordinate Y
Z	Real	Raumkoordinate Z
Phi	Real	Euler-Winkel Phi
Theta	Real	Euler-Winkel Theta
Psi	Real	Euler-Winkel Psi
Velocity	Integer	Geschwindigkeit in Prozent
Acceleration	Integer	Beschleunigung (1 oder 2)

Beschreibung
'MoveToPos' bewegt alle Achsen zur angegebenen Zielposition. Diese Funktion erlaubt eine harmonische Bewegung, da die Geschwindigkeiten automatisch an den pro Achse zurückzulegenden Winkel angepasst werden und somit alle Achsen gleichzeitig das Ziel erreichen. Die übergebene Geschwindigkeit gilt für die Achse, welche die grösste Distanz zurücklegt. Als Parameter müssen die Raumkoordinaten und die Anfahrtswinkel übergeben werden. Zusätzlich kann die Beschleunigung angegeben werden.

MoveToPosLin

MoveToPosLin (X; Y; Z; Phi; Theta; Psi {; LinVelocity})

Parameter	Typ	Beschreibung
X	Real	Raumkoordinate X
Y	Real	Raumkoordinate Y
Z	Real	Raumkoordinate Z
Phi	Real	Euler-Winkel Phi
Theta	Real	Euler-Winkel Theta
Psi	Real	Euler-Winkel Psi
LinVelocity	Integer	Raum-Geschwindigkeit des Tool Center Points in Prozent

Beschreibung
'MoveToPosLin' fährt den Roboter linear von der aktuellen Position zur Zielposition. Die Geschwindigkeit beträgt 'LinVelocity'.

MoveToPosDeg

MoveToPosDeg(Angle1; Angle2; Angle3; Angle4; Angle5; Angle6{; Velocity; Acceleration})

Parameter	Typ	Beschreibung
Angle1	Real	Position der Achse 1 in Grad
Angle2	Real	Position der Achse 2 in Grad

Angle3	Real	Position der Achse 3 in Grad
Angle4	Real	Position der Achse 4 in Grad
Angle5	Real	Position der Achse 5 in Grad
Angle6	Real	Position der Achse 6 in Grad
Velocity	Integer	Geschwindigkeit in Prozent
Acceleration	Integer	Beschleunigung (1 oder 2)

Beschreibung
'MoveToPosDeg' fährt jede Achse zur angegebenen Position. Die Geschwindigkeit der einzelnen Achsen wird, wie beim Befehl 'MoveToPos', so angepasst, dass alle Achsen die Zielposition zur selben Zeit erreichen. Der Parameter 'Velocity' entspricht demnach der Maximalgeschwindigkeit, welche die am schnellsten drehende Achse erreichen soll.

MoveDelta

MoveDelta (DeltaX; DeltaY; DeltaZ{; Velocity; Acceleration})

Parameter	**Typ**	**Beschreibung**
DeltaX	Real	Versatz in X-Richtung
DeltaY	Real	Versatz in Y-Richtung
DeltaZ	Real	Versatz in Z-Richtung
Velocity	Integer	Geschwindigkeit in Prozent
Acceleration	Integer	Beschleunigung (1 oder 2)

Beschreibung
'MoveDelta' bewirkt eine Verschiebung um eine Koordinatendifferenz durch eine einzelne Bewegung. Die Greiferorientierung bleibt unverändert.

Beispiel
MoveDelta(0; 0; 10)

Verschiebung in Z-Richtung um 10mm.

MoveDeltaLin

MoveDeltaLin (DeltaX; DeltaY; DeltaZ{; LinVelocity})

Parameter	**Typ**	**Beschreibung**
DeltaX	Real	Versatz in X-Richtung
DeltaY	Real	Versatz in Y-Richtung
DeltaZ	Real	Versatz in Z-Richtung
LinVelocity	Integer	Raum-Geschwindigkeit des Tool Center Points in Prozent

Beschreibung
Wie der Befehl 'MoveDelta', ausser dass die Position linear angefahren wird.

Beispiel
MoveDeltaLin(0; 0; 10)

Lineare Verschiebung in Z-Richtung um 10mm.

MoveRadial

MoveRadial (Distance {; Velocity; Acceleration})

Parameter	Typ	Beschreibung
Distance	Real	Distanz in radialer Richtung
Velocity	Integer	Geschwindigkeit in Prozent
Acceleration	Integer	Beschleunigung (1 oder 2)

Beschreibung
Unabhängig von den X-Y-Achsenausrichtungen wird der Roboter radial, in Bezug auf die Achse 1 gefahren. Die Distanz kann positiv oder negativ sein. Bei einer positiven Distanz bewegt sich der Tool Center Point vom Koordinatenursprung weg.

Beispiel
MoveRadial (-30)

Fahre 30 mm in Richtung des Koordinatenursprungs.

MoveRadialLin

MoveRadialLin (Distance {; LinVelocity})

Parameter	Typ	Beschreibung
Distance	Real	Distanz in radialer Richtung
LinVelocity	Integer	Raum-Geschwindigkeit des Tool Center Point in Prozent

Beschreibung
Wie der Befehl 'MoveRadial', ausser dass die Position linear angefahren wird.

Beispiel
MoveRadialLin(50)

Fahre linear 50mm in Richtung des Koordinatenursprungs.

MoveInToolDirection

MoveInToolDirection (DeltaX; DeltaY; DeltaZ ; Velocity; Acceleration)

Parameter	Typ	Beschreibung
DeltaX	Real	Distanz in Tool-X-Richtung
DeltaY	Real	Distanz in Tool-Y-Richtung
DeltaZ	Real	Distanz in Tool-Z-Richtung
Velocity	Integer	Geschwindigkeit in Prozent
Acceleration	Integer	Beschleunigung (1 oder 2)

Beschreibung
Der Roboter fährt die übergebene Distanz in Greiferrichtung (Toolrichtung). Die Greiferorientierung bleibt erhalten.

Beispiel
MoveInToolDirection (-10; 0; 0; 30; 1)

Der Roboter fährt mit 30 prozentiger Geschwindigkeit in Greiferrichtung 10mm nach vorne

MoveInToolDirectionLin

MoveInToolDirectionLin (DeltaX; DeltaY; DeltaZ ; LinVelocity)

Parameter	Typ	Beschreibung
DeltaX	Real	Distanz in Tool-X-Richtung
DeltaY	Real	Distanz in Tool-Y-Richtung
DeltaZ	Real	Distanz in Tool-Z-Richtung
LinVelocity	Integer	Raum-Geschwindigkeit des Tool Center Point in Prozent

Beschreibung
Der Roboter fährt linear die übergebene Distanz in Greiferrichtung (Toolrichtung). Die Greiferorientierung bleibt erhalten.

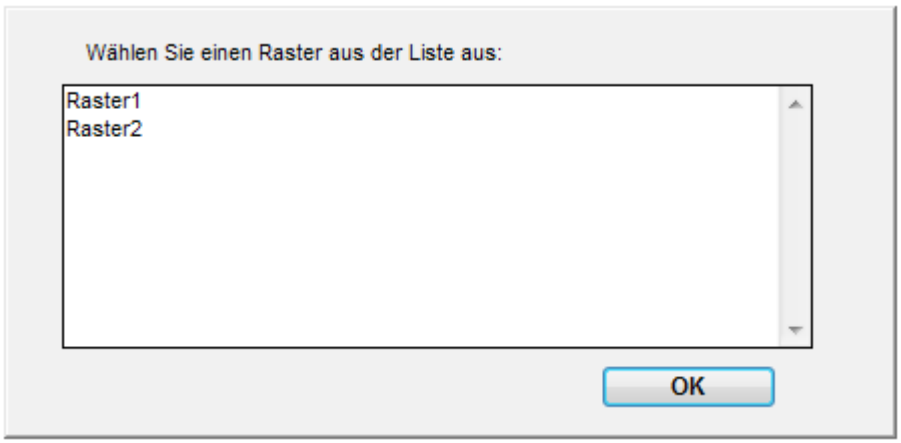
Beispiel
MoveInToolDirectionLin (-10; 0; 0; 30)

Der Roboter fährt linear mit 30 prozentiger Geschwindigkeit in Greiferrichtung 10mm zurück.

2.1.3 Anfahren von Punkten im Raster

Bei den Befehlen in diesem Abschnitt werden die Raster benützt, die im Projekt unter dem Register 'Raster' erstellt worden sind.

Nach einem Doppelklick auf den entsprechenden Befehl erscheint ein neues Fenster. In diesem werden die verfügbaren Raster angezeigt und können ausgewählt werden. Nach der Bestätigung wird der Name des Rasters direkt als Parameter eingefügt.



MoveToGridPoint

MoveToGridPoint (Name; PointNr; Velocity; Acceleration) *

Parameter	Typ	Beschreibung
Name	String	Name des Rasters
PointNr	Integer	Nummer des Punktes im Raster
Velocity	Integer	Geschwindigkeit in Prozent
Acceleration	Integer	Beschleunigung (1 oder 2)

Beschreibung
'MoveToGridPoint' fährt den Roboter im übergebenen Raster zur Position mit der entsprechenden Nummer.
Wird der angegeben Punkt nicht gefunden, erscheint eine Fehlermeldung und das Programm wird beendet. Der Befehl verwendet intern den Befehl 'MoveToPos'.

Beispiel
MoveToGridPoint(“Raster 1“; 1; 100;1)

Der erste Punkt im Raster „Raster 1“ wird angefahren.

MoveToGridPointDelta

MoveToGridPointDelta (“Name“; PointNr; DeltaX; DeltaY; DeltaZ; Velocity; Acceleration) *

Parameter	Typ	Beschreibung
Name	String	Name des Rasters
PointNr	Integer	Nummer des Punktes im Raster
DeltaX	Real	Versatz in X-Richtung
DeltaY	Real	Versatz in Y-Richtung
DeltaZ	Real	Versatz in Z-Richtung
Velocity	Integer	Geschwindigkeit in Prozent
Acceleration	Integer	Beschleunigung (1 oder 2)

Beschreibung
Gleich wie der Befehl 'MoveToGridPoint', ausser dass noch ein Versatz zum Rasterpunkt angegeben werden kann.

MoveToGridPointLin

MoveToGridPointLin (“Name“; PointNr; LinVelocity) *

Parameter	Typ	Beschreibung
Name	String	Name des Rasters
PointNr	Integer	Nummer des Punktes im Raster
LinVelocity	Integer	Raum-Geschwindigkeit des Tool Center Points in Prozent

Beschreibung
Lineare Bewegung zum Rasterpunkt. Der Befehl verwendet intern den Befehl 'MoveToPosLin'.

MoveToGridPointLinDelta

MoveToGridPointLinDelta (“Name“; PointNr; DeltaX; DeltaY; DeltaZ; LinVelocity) *

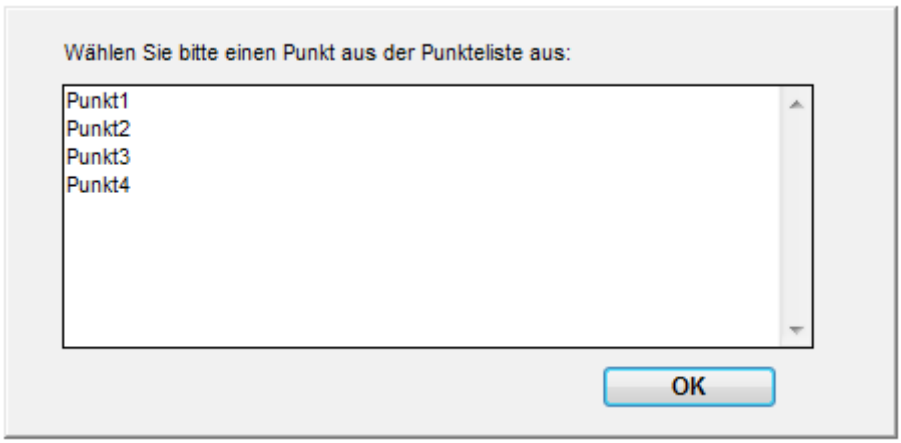
Parameter	Typ	Beschreibung
Name	String	Name des Rasters
PointNr	Integer	Nummer des Punktes im Raster
DeltaX	Real	Versatz in X-Richtung
DeltaY	Real	Versatz in Y-Richtung
DeltaZ	Real	Versatz in Z-Richtung
LinVelocity	Integer	Raum-Geschwindigkeit des Tool Center Points in Prozent

Beschreibung
Eine lineare Bewegung zum Rasterpunkt mit dem Versatz (DeltaX; DeltaY; DeltaZ).

2.1.4 Anfahren von Punkten im Raum

Bei den Befehlen in diesem Abschnitt werden die Punkte benützt, die im Projekt unter dem Register 'Punkte' gespeichert sind.

Nach einem Doppelklick auf den entsprechenden Befehl erscheint ein neues Fenster. In diesem können die verfügbaren Punkte ausgewählt werden. Der ausgewählte Punkt wird direkt als Parameter in den Befehl integriert.



MoveToPoint

MoveToPoint (“Name“; Velocity; Acceleration) *

Parameter	**Typ**	**Beschreibung**
Name	String	Name des Punktes
Velocity	Integer	Geschwindigkeit in Prozent
Acceleration	Integer	Beschleunigung (1 oder 2)

Beschreibung
'MoveToPoint' fährt den Roboter zum Punkt, der als Parameter übergeben wurde. Wird der angegebene Punkt nicht gefunden, erscheint eine Fehlermeldung und das Programm wird beendet. Der Befehl verwendet intern den Befehl 'MoveToPos'.

Beispiel
MoveToPoint(“PunktOberhalbStift“; 80; 1)

MoveToPointDelta

MoveToPointDelta (“Name“; DeltaX; DeltaY; DeltaZ; Velocity; Acceleration)

Parameter	Typ	Beschreibung
Name	String	Name des Punktes
DeltaX	Real	Versatz in X-Richtung
DeltaY	Real	Versatz in Y-Richtung
DeltaZ	Real	Versatz in Z-Richtung
Velocity	Integer	Geschwindigkeit in Prozent
Acceleration	Integer	Beschleunigung (1 oder 2)

Beschreibung
Der übergebene Punkt wird mit dem Versatz (DeltaX; DeltaY; DeltaZ) angefahren. 'MoveToPointDelta' eignet sich, um in Kombination mit einer Linearbahn, einen Zielpunkt exakt anzufahren.

Beispiel
MoveToPointDelta(“PunktOberhalbStift“; 0; 0; 30; 80;1)

Fahre 30mm über den Punkt 'PunktOberhalbStift'.

MoveToPointDeg

MoveToPointDeg (“Name“; Velocity; Acceleration) *

Parameter	Typ	Beschreibung
Name	String	Name des Punktes
Velocity	Integer	Geschwindigkeit in Prozent
Acceleration	Integer	Beschleunigung (1 oder 2)

Beschreibung
Dieser Befehl fährt den Punkt genau so an, wie er geteacht wurde. Er verwendet intern den Befehl 'MoveToPosDeg'. Der Unterschied zum Befehl 'MoveToPoint' ist, dass Achse 6 zu der Position fährt, die in der Punkteliste gespeichert ist und nicht die aktuelle Position behält.

Beispiel
MoveToPointDeg(“PunktOberhalbStift“; 80; 1)

MoveToPointLin

MoveToPointLin (“Name“; LinVelocity) *

Parameter	Typ	Beschreibung
Name	String	Name des Punktes
LinVelocity	Integer	Raum-Geschwindigkeit des Tool Center Points in Prozent

Beschreibung
'MoveToPointLin' fährt den Roboter linear zum angegebenen Punkt 'Name'.

Beispiel
MoveToPointLin(“PunktStift“;80)

MoveToPointLinDelta

MoveToPointLinDelta (“Name“; DeltaX; DeltaY; DeltaZ; LinVelocity)

Parameter	Typ	Beschreibung
Name	String	Name des Punktes
DeltaX	Real	Versatz in X-Richtung
DeltaY	Real	Versatz in Y-Richtung
DeltaZ	Real	Versatz in Z-Richtung
LinVelocity	Integer	Raum-Geschwindigkeit des Tool Center Points in Prozent

Beschreibung
'Name' wird linear mit dem Abstand ('DeltaX'; 'DeltaY'; 'DeltaZ') angefahren.

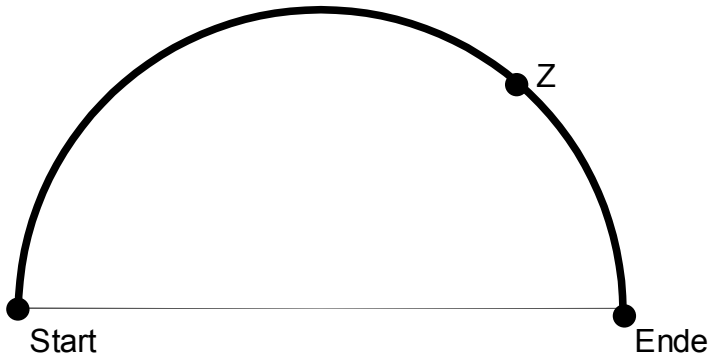
Beispiel
MoveToPointLinDelta(“PunktOberhalbStift“; 0; 0; -30; 80)

MoveToPointArc

MoveToPointArc (“StartPoint“; “EndPoint“; “HelpPoint“; LinVelocity) *

Parameter	Typ	Beschreibung
StartPoint	String	Name des Startpunkts
EndPoint	String	Name des Endpunkts
HelpPoint	String	Name des Hilfspunkts
LinVelocity	Integer	Raum-Geschwindigkeit des Tool Center Points in Prozent

Beschreibung
'MoveToPointArc' fährt eine Kreisbahn von 'StartPoint' über 'HelpPoint' zu 'EndPoint'.
Befindet sich der Roboter nicht am Startpunkt, wird dieser zuerst angefahren.



Beispiel
MoveToPointArc (“Start“; “Ende“; “Z“; 60)

2.1.5 Abfahren der Wege

MoveAlongPath

MoveAlongPath (“Name“{; PathVelocity{; Direction}}) *

Parameter	Typ	Beschreibung
Name	String	Name des Weges
PathVelocity	Integer	Geschwindigkeit im Verhältnis zur Aufzeichnung
Direction	Integer	Abfahr-Richtung (-1 oder 1)

Beschreibung
Mit 'MoveAlongPath' kann ein aufgezeichneter Weg abgefahren werden. Die Geschwindigkeit ist davon abhängig, wie schnell der Weg aufgezeichnet wurde. Geschwindigkeit 100 (%) heisst, dass der Roboter den Weg gleich schnell abfährt wie aufgenommen. Die Geschwindigkeit kann beliebig angepasst werden (mehr als 100 möglich). Bei zu hohen Geschwindigkeiten, wird der Weg unsauber abgefahren. Daher sollte vorsichtig ans Limit gegangen werden.
Im dritten Parameter (optional) wird angegeben, in welcher Richtung der Weg abgefahren werden soll. Wird 1 oder nichts übergeben, fährt der Roboter in die aufgezeichnete Richtung. Bei -1 in die umgekehrte Richtung.
Befindet sich der Roboter nicht am Startpunkt des Weges, wird dieser zuerst angefahren.

Beispiel
MoveAlongPath("TestWeg";400;-1)

'TestWeg' wird mit 4-facher Geschwindigkeit (im Verhältnis zur Aufzeichnung) abgefahren. Die Richtung ist der Aufzeichnungsrichtung entgegengesetzt.

2.1.6 Abfahren der Trajektorien

MoveTrajectory

MoveTrajectory (“Name“{; Velocity{; Direction}}) *

Parameter	Typ	Beschreibung
Name	String	Name der Trajektorie
Velocity	Integer	Geschwindigkeit in Prozent
Direction	Integer	Abfahr-Richtung (-1 oder 1)

Beschreibung
Mit 'MoveTrajectory' kann eine Trajektorie abgefahren werden. Die Geschwindigkeit ist abhängig von den projektseitig eingestellten Werten (jeder Abschnitt kann seine eigene Geschwindigkeit haben). Eine Geschwindigkeit 100 (%) heisst, dass der Roboter die Trajektorie mit den übergebenen Werten abfährt. Die Geschwindigkeit kann beliebig angepasst werden. Falls die Geschwindigkeit nicht gesetzt wird, wird 100 als Standard genommen.
Mit 'Direction' (optional) kann die Richtung, in der die Trajektorie abgefahren werden soll, angegeben werden. 1 oder nichts bedeutet in die aufgezeichnete und -1 in die umgekehrte Richtung.
Befindet sich der Roboter nicht am Startpunkt der Trajektorie, wird dieser zuerst angefahren.

Beispiel
MoveTrajectory("TC1")

Dieses Beispiel fährt die Trajektorie „TC1“ ab.

2.1.7 Lesen und Speichern

Punkte

GetPoint

GetPoint (“Name“; ->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi)

Parameter	Typ	Beschreibung
Name	String	Name des Punkts, der ausgelesen werden soll
->X	Zeiger auf Real	X-Koordinate
->Y	Zeiger auf Real	Y-Koordinate
->Z	Zeiger auf Real	Z-Koordinate
->Phi	Zeiger auf Real	Euler-Winkel Phi
->Theta	Zeiger auf Real	Euler-Winkel Theta
->Psi	Zeiger auf Real	Euler-Winkel Psi

Beschreibung
'GetPoint' liest die Raumkoordinaten und Winkel von einem Punkt aus dem Projekt aus und speichert diese in den, als Zeiger übergebenen, Variablen ab.

Beispiel
GetPoint("Start";->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi)
SavePoint("Start_copy"; X; Y; Z; Phi; Theta; Psi)

Die Werte vom Punkt 'Start' werden ausgelesen und im Punkt 'Start_copy' gespeichert.

GetPointDeg

GetPointDeg (“Name“; ->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6)

Parameter	Typ	Beschreibung
Name	String	Name vom Punkt
->Ang1	Zeiger auf Real	Position der Achse 1
->Ang2	Zeiger auf Real	Position der Achse 2
->Ang3	Zeiger auf Real	Position der Achse 3
->Ang4	Zeiger auf Real	Position der Achse 4
->Ang5	Zeiger auf Real	Position der Achse 5
->Ang6	Zeiger auf Real	Position der Achse 6

Beschreibung
'GetPointDeg' liest die Position der Achsen von einem Punkt aus dem Projekt aus.

Beispiel
GetPointDeg("Start"; ->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6)
SavePointDeg("Start_copy2"; Ang1; Ang2; Ang3; Ang4; Ang5; Ang6)

Die Werte vom Punkt 'Start' werden ausgelesen und im Punkt 'Start_copy' gespeichert.

SavePoint

SavePoint (“Name“; X; Y; Z; Phi; Theta; Psi) -> Boolean

Parameter	Typ	Beschreibung
Name	String	Name vom Punkt
X	Real	X-Koordinate
Y	Real	Y-Koordinate
Z	Real	Z-Koordinate
Phi	Real	Euler-Winkel Phi
Theta	Real	Euler-Winkel Theta
Psi	Real	Euler-Winkel Psi
Funktionsergebnis:	Boolean	erfolgreiche Speicherung

Beschreibung
'SavePoint' speichert einen Punkt im Projekt ab. Falls der Punkt schon vorhanden ist, werden die Werte überschrieben, sonst wird der Punkt neu erstellt.
Zu den Koordinaten und Winkeln werden die entsprechenden Positionen der Achsen berechnet und abgespeichert.
Falls der Punkt ausserhalb des Arbeitsbereiches vom Roboter liegt, wird er nicht gespeichert und 'false' wird zurückgegeben.

Beispiel
GetPoint("Start"; ->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi)
SavePoint("Start_copy"; X; Y; Z; Phi; Theta; Psi)

Die Werte vom Punkt 'Start' werden ausgelesen und im Punkt 'Start_copy' gespeichert.

Hinweis
Bei der Berechnung der Positionen aller Achsen wird die erste Lösung ausgewählt. Es kann sein, dass sich der Roboter dadurch anders positioniert als gewünscht. Sollte dies ein Problem darstellen, muss der Befehl 'SavePointDeg' verwendet werden, bei dem die eindeutigen Positionen der Achsen übernommen werden.

SavePointDeg

SavePointDeg (“Name“; Ang1; Ang2; Ang3; Ang4; Ang5; Ang6) -> Boolean

Parameter	Typ	Beschreibung
Name	String	Name vom Punkt
Ang1	Integer	Position der Achse 1
Ang2	Integer	Position der Achse 2
Ang3	Integer	Position der Achse 3
Ang4	Integer	Position der Achse 4
Ang5	Integer	Position der Achse 5
Ang6	Integer	Position der Achse 6
Funktionsergebnis:	Boolean	erfolgreiche Speicherung

Beschreibung
'SavePointDeg' speichert einen Punkt im Projekt ab. Falls der Punkt schon vorhanden ist, werden die Werte überschrieben, andernfalls wird der Punkt neu erstellt.
Zu den Positionen der Achsen werden die Raumkoordinaten und Winkel berechnet und abgespeichert.
Falls der Punkt ausserhalb des Arbeitsbereiches vom Roboter liegt, wird er nicht gespeichert und 'false' wird zurückgegeben.

Beispiel
GetPointDeg("Start"; ->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6)
SavePointDeg("Start_copy2"; Ang1; Ang2; Ang3; Ang4; Ang5; Ang6)

Die Werte vom Punkt 'Start' werden ausgelesen und im Punkt 'Start_copy' gespeichert.

Raster

GetGridPoint

GetGridPoint (“Name“; PointNr; ->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi)

Parameter	Typ	Beschreibung
Name	String	Name vom Raster
PointNr	Integer	Nummer des Punktes
->X	Zeiger auf Real	X-Koordinate
->Y	Zeiger auf Real	Y-Koordinate
->Z	Zeiger auf Real	Z-Koordinate
->Phi	Zeiger auf Real	Euler-Winkel Phi
->Theta	Zeiger auf Real	Euler-Winkel Theta
->Psi	Zeiger auf Real	Euler-Winkel Psi

Beschreibung
'GetGridPoint' liest die Raumkoordinaten und Winkel von einem Rasterpunkt aus dem Projekt aus. Beim Parameter 'Name' muss der Name des Rasters und bei 'PointNr' die Nummer des Rasterpunkts angegeben werden, der ausgelesen werden soll.

Beispiel
GetGridPoint("Grid1"; 1; ->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi)
SaveGridPoint("Grid1"; 2; X; Y; Z; Phi; Theta; Psi)

GetGridPointDeg

GetGridPointDeg (“Name“; PointNr; ->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6)

Parameter	Typ	Beschreibung
Name	String	Name vom Raster
PointNr	Integer	Nummer des Punktes
->Ang1	Zeiger auf Real	Position der Achse 1
->Ang2	Zeiger auf Real	Position der Achse 2
->Ang3	Zeiger auf Real	Position der Achse 3
->Ang4	Zeiger auf Real	Position der Achse 4
->Ang5	Zeiger auf Real	Position der Achse 5
->Ang6	Zeiger auf Real	Position der Achse 6

Beschreibung
'GetGridPointDeg' liest die Positionen der Achsen von einem Rasterpunkt aus dem Projekt aus. Beim Parameter 'Name' muss der Name des Rasters und bei 'PointNr' die Nummer des Rasterpunkts angegeben werden.

Beispiel
GetGridPointDeg("Grid1"; 1; ->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6)
SaveGridPointDeg("Grid1"; 2; Ang1; Ang2; Ang3; Ang4; Ang5; Ang6)

Die Werte vom Rasterpunkt '1' des Rasters 'Grid1' werden ausgelesen und im Rasterpunkt '2' gespeichert.

SaveGridPoint

SaveGridPoint (“Name“; PointNr; X; Y; Z; Phi; Theta; Psi) -> Boolean

Parameter	Typ	Beschreibung
Name	String	Name des Rasters
PointNr	Integer	Nummer des Punkts
X	Real	X-Koordinate
Y	Real	Y-Koordinate
Z	Real	Z-Koordinate
Phi	Real	Euler-Winkel Phi
Theta	Real	Euler-Winkel Theta
Psi	Real	Euler-Winkel Psi
Funktionsergebnis:	Boolean	erfolgreiche Speicherung

Beschreibung
'SaveGridPoint' speichert einen Rasterpunkt im Projekt ab. Es können nur Punkte überschrieben, aber keine neuen erstellt werden.
Zu den Koordinaten und Winkeln werden die entsprechenden Positionen der Achsen berechnet und abgespeichert.
Falls der Rasterpunkt ausserhalb des Arbeitsbereiches des Roboters liegt, wird er nicht gespeichert und 'false' wird zurückgegeben.

Beispiel
GetGridPoint("Grid1"; 1; ->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi)
SaveGridPoint("Grid1"; 2; X; Y; Z; Phi; Theta; Psi)

Die Werte vom Rasterpunkt '1' des Rasters 'Grid1' werden ausgelesen und im Rasterpunkt '2' gespeichert.

Hinweis
Bei der Berechnung der Positionen der Achsen wird die erste Lösung ausgewählt. Es kann sein, dass sich der Roboter dadurch anders positioniert als gewünscht. Sollte dies ein Problem darstellen, muss der Befehl 'SaveGridPointDeg' verwendet werden, bei dem die eindeutigen Positionen der Achsen übernommen werden.

SaveGridPointDeg

SaveGridPointDeg(“Name“; PointNr; Ang1; Ang2; Ang3; Ang4; Ang5; Ang6)->Boolean

Parameter	Typ	Beschreibung
Name	String	Name vom Raster
PointNr	Integer	Nummer des Punktes
Ang1	Real	Position der Achse 1
Ang2	Real	Position der Achse 2
Ang3	Real	Position der Achse 3
Ang4	Real	Position der Achse 4
Ang5	Real	Position der Achse 5
Ang6	Real	Position der Achse 6
Funktionsergebnis:	Boolean	erfolgreiche Speicherung

Beschreibung
'SaveGridPointDeg' speichert einen Rasterpunkt im Projekt ab. Es können nur Punkte überschrieben, aber keine neuen erstellt werden.
Zu den Positionen der Achsen werden die Raumkoordinaten und Winkel berechnet und abgespeichert.
Falls der Punkt ausserhalb des Arbeitsbereiches vom Roboter liegt, wird er nicht gespeichert und die Funktion liefert 'false' zurück.

Beispiel
GetGridPointDeg("Grid1"; 1; ->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6)
SaveGridPointDeg("Grid1"; 2; Ang1; Ang2; Ang3; Ang4; Ang5; Ang6)

Die Werte vom Rasterpunkt '1' des Rasters 'Grid1' werden ausgelesen und im Rasterpunkt '2' gespeichert.

Benutzervariablen

GetIntVar

GetIntVar (“Name“) -> Integer

Parameter	Typ	Beschreibung
Name	String	Name der Variable
Funktionsergebnis:	Integer	Wert der Variable

Beschreibung
'GetIntVar' gibt den Wert einer Integer-Benutzervariablen zurück.

Beispiel
VarInt1 := GetIntVar("Var")

Der Wert der Integer-Benutzervariable 'Var' wird in die Variable 'VarInt1' geschrieben.

GetRealVar

GetRealVar (“Name“) -> Real

Parameter	Typ	Beschreibung
Name	String	Name der Variable
Funktionsergebnis:	Real	Wert der Variable

Beschreibung
'GetRealVar' gibt den Real-Wert einer Benutzervariablen zurück.

Beispiel
VarReal1 := GetRealVar("Var")

Der Wert der Real-Benutzervariable 'Var' wird in die Variable 'VarReal1' geschrieben.

GetStrVar

GetStrVar (“Name“) -> String

Parameter	Typ	Beschreibung
Name	String	Name der Variable
Funktionsergebnis	String	Wert der Variable

Beschreibung
'GetStrVar' gibt den Wert einer String-Benutzervariablen zurück.

Beispiel
VarStr1 := GetStrVar("Var")

Der Wert der String-Benutzervariable 'Var' wird in die Variable 'VarStr1' geschrieben.

SetIntVar

SetIntVar (“Name“; Value)

Parameter	Typ	Beschreibung
Name	String	Name der Variable
Value	Integer	Wert der Variable

Beschreibung
'SetIntVar' schreibt eine Wert in eine Integer-Benutzervariable.

Beispiel
SetIntVar("Var", 2)

Schreibt '2' in die Integer-Benutzervariable 'Var'.

SetRealVar

SetRealVar (“Name“; Value)

Parameter	Typ	Beschreibung
Name	String	Name der Variable
Value	Real	Wert der Variable

Beschreibung
'SetRealVar' schreibt einen Wert in eine Real-Benutzervariable.

Beispiel
SetRealVar("Var", 3.14)

Schreibt '3.14' in die Real-Benutzervariable 'Var'.

SetStrVar

SetStrVar (“Name“; Value)

Parameter	Typ	Beschreibung
Name	String	Name der Variable
Value	String	Wert der Variable

Beschreibung
'SetStrVar' schreibt einen Wert in eine String-Benutzervariable.

Beispiel
SetStrVar("Var"; “Beispiel“)

Schreibt 'Beispiel' in die String-Benutzervariable 'Var'.

2.2 Zustandsabfragen

ReadCoordinates

ReadCoordinates (->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi)

Parameter	Typ	Beschreibung
X	Zeiger auf Real	Raumkoordinate X
Y	Zeiger auf Real	Raumkoordinate Y
Z	Zeiger auf Real	Raumkoordinate Z
Phi	Zeiger auf Real	Euler-Winkel Phi
Theta	Zeiger auf Real	Euler-Winkel Theta
Psi	Zeiger auf Real	Euler-Winkel Psi

Beschreibung
'ReadCoordinates' liefert die Raumkoordinaten und Winkel der aktuellen Position zurück.

Beispiel
ReadCoordinates(->X; ->Y; ->Z; ->Phi; ->Theta; ->Psi)
MoveToPos (X; Y; Z; Phi; Theta; Psi + 20; 100; 1)

Liest die aktuelle Position aus und fährt die Position mit einem um 20° veränderten Winkel Psi an.

ReadAxisPos

ReadAxisPos (AxisNr) * -> Integer

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 – 6)
Funktionsergebnis:	Integer	Position der Achse

Beschreibung
'ReadAxisPos' liefert die aktuelle Position der entsprechenden Achse.

Beispiel
VarInt1 := ReadAxisPos (3)

Die aktuelle Position der Achse 3 in die Variable 'VarInt1' speichern.

ReadPositions

ReadPositions (->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6)

Parameter	Typ	Beschreibung
->Ang1	Zeiger auf Real	Position der Achse 1
->Ang2	Zeiger auf Real	Position der Achse 2
->Ang3	Zeiger auf Real	Position der Achse 3
->Ang4	Zeiger auf Real	Position der Achse 4
->Ang5	Zeiger auf Real	Position der Achse 5
->Ang6	Zeiger auf Real	Position der Achse 6

Beschreibung
'ReadPositions' liefert die Raumkoordinaten und Winkel der aktuellen Position zurück.

Beispiel
ReadPositions (->Ang1; ->Ang2; ->Ang3; ->Ang4; ->Ang5; ->Ang6)
MoveToPosDeg (Ang1; Ang2; Ang3; Ang4; Ang5; Ang6 + 10; 80; 1)

Liest die aktuelle Position aus und fährt diese Position mit einem Unterschied von 10° in Achse 6 an.

ReadAxisState

ReadAxisState (AxisNr) * -> Integer

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 – 6)
Funktionsergebnis:	Integer	Status der Achse

Beschreibung
'ReadAxisState' liefert den aktuellen Status der Achse.

0 = off (ausgeschaltet)
8 = in gewünschter Position (Roboter fixiert)
24 = im “normalen Bewegungsstatus“; versucht die gewünschte Position zu erreichen
40 = wegen einer Kollision gestoppt
88 = die Linearbewegung ist beendet
128 = ungültige Achsendaten (interne Kommunikationsprobleme)

ReadMaxVel

ReadMaxVel (AxisNr{; ->VarInt2; ->VarInt3}) -> Integer

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 - 6)
->VarInt2	Zeiger auf Integer	maximale positive Geschwindigkeit des Motors
->VarInt3	Zeiger auf Integer	maximale negative Geschwindigkeit des Motors
Funktionsergebnis:	Integer	maximale positive Geschwindigkeit des Motors

Beschreibung
'ReadMaxVel' liefert die maximale positive Geschwindigkeit, welche die entsprechende Achse erreichen kann. Will man die positive und negative Geschwindigkeit ermitteln, dann müssen zwei Benutzervariablen als Zeiger übergeben werden. Die Variablen 'VarInt2' und 'VarInt3' enthalten dann die Geschwindigkeiten.

Beispiele
VarInt1 := ReadMaxVel (3)
Die maximale positive Geschwindigkeit der Achse 3 ermitteln.

ReadMaxVel (2; ->VarInt1; ->VarInt2)
Die positive und negative Geschwindigkeit der Achse 2 ermitteln.

ReadMaxPWM

ReadMaxPWM (AxisNr{; ->VarInt2; ->VarInt3}) -> Integer

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 - 6)
->VarInt2	Zeiger auf Integer	maximale positive PWM der Achse
->VarInt3	Zeiger auf Integer	maximale negative PWM der Achse
Funktionsergebnis:	Integer	maximale positive PWM des Motors

Beschreibung
'ReadMaxPWM' liefert die maximale positive PWM, welche die entsprechende Achse erreichen kann. Will man die positive und negative PWM ermitteln, dann müssen zwei Benutzervariablen als Zeiger übergeben werden. Die Variablen 'VarInt2' und 'VarInt3' enthalten dann die PWM.

Beispiele
VarInt1 := ReadMaxPWM (6)
Den maximale positive PWM des Motors 6 ermitteln.

ReadMaxPWM (5; ->VarInt1; ->VarInt2)
Den positiven und negativen PWM-Wert der Achse 5 ermitteln.

ReadMaxAcc

ReadMaxAcc (AxisNr) -> Integer

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 - 6)
Funktionsergebnis:	Integer	Beschleunigung der Achse

Beschreibung
Liefert die maximale Beschleunigung, welche die entsprechende Achse erreichen kann.

Beispiel
VarInt1 := ReadMaxAcc (3)
Die aktuelle Beschleunigung der Achse 3 in die Variable 'VarInt1' speichern.

ReadCollisionPosLimit

ReadCollisionPosLimit (AxisNr) -> Integer

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 – 6)
Funktionsergebnis:	Integer	Kollisionslimit der Achse

Beschreibung
'ReadCollisionPosLimit' liefert das Kollisionslimit für eine Bewegung der entsprechenden Achse.

Beispiel
VarInt1 := ReadCollisionPosLimit (1)

Das Kollisionslimit der Achse 1 in die Variable 'VarInt1' speichern.

ReadCollisionPosLimitLin

ReadCollisionPosLimitLin (AxisNr) -> Integer

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 – 6)
Funktionsergebnis:	Integer	Kollisionslimit der Achse

Beschreibung
'ReadCollisionPosLimitLin' liefert das Kollisionslimit für eine lineare Bewegung der entsprechenden Achse.

Beispiel
VarInt1 := ReadCollisionPosLimitLin (2)

Das Kollisionslimit der Achse 2 in die Variable 'VarInt1' speichern.

ReadK (AxisNr; ->VarInt1; ->VarInt2; ->VarInt3)

ReadK (Motornummer; ->VarInt1; ->VarInt2; ->VarInt3)

Parameter	Typ	Beschreibung
AxisNr	Integer	Achse (1 - 6)
->VarInt1	Zeiger auf Integer	Kspeed
->VarInt2	Zeiger auf Integer	Kpos
->VarInt3	Zeiger auf Integer	Ki

Beschreibung
'ReadK' liefert die drei Achscontroller-Konstanten (Kspeed, Kpos, Ki) für die entsprechende Achse. Es müssen drei Benutzervariablen als Zeiger übergeben werden. Die Variablen 'VarInt1', 'VarInt2' und 'VarInt3' enthalten dann die Werte.

Beispiel
ReadK (3; ->VarInt1; ->VarInt2; ->VarInt3)

Die Achscontroller-Konstanten der Achse 3 ermitteln.

ReadSensor

ReadSensor(SensorNr) -> Integer

Parameter	Typ	Beschreibung
SensorNr	Integer	Nummer des Sensors
Funktionsergebnis:	Integer	Sensorwert (0-255)

Beschreibung
'ReadSensor' liefert den Wert des angegebenen Sensors.

Beispiel
VarInt1 := ReadSensor (1)

Liefert den Wert von Sensor 1.

2.3 Schnittstellen

2.3.1 National Instruments IO Card

NI_ReadInput

NI_ReadInput (InputNr) -> Integer

Parameter	Typ	Beschreibung
InputNr	Integer	PIN-Nummer
Funktionsergebnis:	Integer	Zustand 0 oder 1

Beschreibung
'NI_ReadInput' gibt den Zustand des übergebenen Inputs (1 bis 32) zurück. Wenn der entsprechende Input gesetzt ist, wird eine 1, ansonsten eine 0 zurückgegeben.

Beispiel
VarInt1 := NI_ReadInput(1)

Liefert den Wert des ersten Inputs.

NI_SetOutput

NI_SetOutput (OutputNr; Value)

Parameter	Typ	Beschreibung
OutputNr	Integer	PIN-Nummer
Value	Integer	Zustand 0 oder 1

Beschreibung
Mit 'NI_SetOutput' kann ein Output (1 bis 32) ein- oder ausgeschaltet werden. Wird im zweiten Parameter eine 1 übergeben, wird der Output eingeschaltet, sonst ausgeschaltet.

Beispiel
NI_SetOutput(1;0)

Output 1 wird ausgeschaltet.

NI_ReadBank(BankNr)

NI_ReadBank (BankNr) -> Integer

Parameter	Typ	Beschreibung
BankNr	Integer	Bank-Nummer
Funktionsergebnis:	Integer	Zustand der Pins der entsprechenden Bank

Beschreibung
Durch 'NI_ReadBank' wird eine Bank (Pin 1 bis 8) gelesen und eine Zahl zwischen 0 und 255 zurückgegeben. Sie setzt sich durch die Pins bitweise zu einem Byte zusammen. Dadurch kann man relativ einfach mit anderen Systemen, die über diese Karte miteinander verbunden sind, kommunizieren.

Beispiel
VarInt1 := NI_ReadBank(1)

Die erste Bank wird gelesen und das Ergebnis in Variable 'VarInt1' gespeichert.

NI_SetBank

NI_SetBank(BankNr; Value)

Parameter	Typ	Beschreibung
BankNr	Integer	Bank-Nummer
Value	Integer	Wert, den die Bank repräsentieren soll

Beschreibung
Mit 'NI_SetBank' kann 'Value' auf einer Output-Bank ausgegeben werden.

Beispiel
NI_SetBank(1; 255)

255 wird auf Bank 1 ausgegeben (alle Pins eingeschaltet).

2.3.2 Datei-Kommunikation

WriteFile

WriteFile (“FileName“; Value) -> Boolean

Parameter	Typ	Beschreibung
FileName	String	Pfad und Dateiname
Value	String	Text, der in die Datei geschrieben werden soll
Funktionsergebnis	Boolean	erfolgreiches Schreiben

Beschreibung
'WriteFile' schreibt den übergebenen Text in eine Datei. Wenn die Datei nicht erstellt werden konnte, wird 'false' zurückgeliefert.

Hinweis
Wenn bei 'FileName' ein leerer String übergeben wird, erscheint ein Speichern-Dialog.

Beispiel
```
If (WriteFile("C:\test.txt"; "Text in File"))
        Alert(“Schreiben war erfolgreich“)
Else
        Alert(“Schreiben war nicht erfolgreich“)
End If
```

Erstellt eine Datei mit dem Inhalt “Text in File“. Danach wird eine Meldung ausgegeben, ob die Aktion erfolgreich war oder nicht.

AddToFile

AddToFile (FileName; Value{; CreateDocument}) -> Boolean

Parameter	Typ	Beschreibung
FileName	String	Pfad und Dateiname
Value	String	Text, der in die Datei geschrieben werden soll
CreateDocument	Boolean	Dokument erstellen, wenn nicht vorhanden
Funktionsergebnis:	Boolean	erfolgreiches Schreiben

Beschreibung
'AddToFile' fügt einer Datei einen Text hinzu. Wenn der Text nicht hinzugefügt werden konnte, wird 'false' zurückgeliefert. Wird für Parameter 'CreateDocument' der Wert 'true' übergeben, wird die Datei erstellt, falls sie noch nicht existiert.

Beispiel
```
AddToFile("C:\test.txt"; char(13) + char(10) + "added Text")
```

Fügt der Datei 'test.txt' einen Zeilenumbruch und den Text “added Text“ hinzu.

ReadFile

ReadFile (“FileName“; DocumentPosition; NumChar{; StopChar}) -> String

Parameter	Typ	Beschreibung
FileName	String	Pfad und Dateiname
DocumentPosition	LongInteger	Position ab der gelesen werden soll
NumChar	LongInteger	Anzahl Zeichen
StopChar	Char	Zeichen bis zu dem gelesen werden soll
Funktionsergebnis	String	ausgelesener Text

Beschreibung
'ReadFile' öffnet eine Datei und liest eine Zeichenkette aus, die als Ergebnis zurückgeliefert wird.
Mit 'DocumentPosition' wird die Position angegeben, bei der begonnen werden soll. Mit 0 wird am Dateianfang begonnen.
Mit 'NumChar' wird die Anzahl Zeichen übergeben, die ausgelesen werden sollen. Wenn für NumChar 0 übergeben wird, muss beim Parameter 'StopChar' das Zeichen angegeben werden, bis zu dem gelesen werden soll.

Beispiele
Alert (ReadFile ("C:\test.txt"; 0; 5))
Liest aus der Datei 'test.txt' die ersten 5 Zeichen aus.

Alert(ReadFile("C:\test.txt";0;“A“))
Liest aus der Datei 'test.txt' alles bis zum Zeichen “A“ aus.

2.3.3 Serielle Kommunikation

SetComport

SetComport (ComNr)

Parameter	Typ	Beschreibung
ComNr	Integer	Nummer des Com-Ports

Beschreibung
Mit 'SetComport' wird der übergebene ComPort geöffnet.

Beispiel
SetComport (1)

ComPort 1 wird geöffnet.

SendSerial

SendSerial (VarStr1)

Parameter	Typ	Beschreibung
VarStr1	String	Text, der gesendet werden soll

Beschreibung
Mit 'SendSerial' werden Daten an den aktuell geöffneten ComPort gesendet. Es kann direkt ein Wert übergeben werden, z.B: SendSerial (“Das ist ein String“) oder eine Variable verwendet werden, z.B: SendSerial(VarStr1). Der ganze Inhalt im Buffer wird zuvor gelöscht.

Beispiel
SetComport (1)
SendSerial (“Test“)

Die Zeichenkette “Test“ wird an den zuvor geöffneten ComPort gesendet.

ReceiveSerial

ReceiveSerial -> Text

Parameter	Typ	Beschreibung
Funktionsergebnis	Text	Text

Beschreibung
Mit ReceiveSerial können Daten vom aktuellen ComPort empfangen werden. Die empfangenen Daten können in einer Variable vom Typ String oder Text gespeichert werden.

Beispiel
SetComport (2)
SendSerial (“RoboterBereit“)
Wait (3000)
VarStr1 := ReceiveSerial
SetComport (1)
//Die Variable VarStr1 beinhaltet die X,Y,Z-Position des gefundenen Objektes.

Empfangen und Senden über den seriellen Port. Der Roboter ist an COM1, eine 3D-Kamera (als Beispiel) an COM2 angeschlossen. Sobald der Roboter an der Startposition ist, wird COM2 geöffnet und das Startsignal geschickt, damit die Kamera eine Messung starten kann. Nach einer Pause von drei Sekunden wird der serielle Port gelesen, weil die Kamera die Position eines gefundenen Objekts zurückgegeben hat. Nach Beendigung der Kommunikation zwischen Roboter und Kamera wird der vorherige ComPort wieder gesetzt.

2.3.4 TCP/IP

TCP_IP_Connect

TCP_IP_Connect (“Hostname“; Portnumber{; Timeout}) * -> Integer

Parameter	Typ	Beschreibung
Hostname	String	Name oder IP des Hosts
Portnumber	Integer	Port des Hosts
Timeout	Integer	Timeout in Sekunden
Funktionsergebnis:	Integer	TCP-Session-ID

Beschreibung
Mit 'TCP_IP_Connect' kann mit dem übergebenen Host eine Verbindung aufgebaut werden. Die Funktion gibt die Session-ID zurück, die beim späteren Senden und Empfangen von Daten wieder gebraucht wird. Katana4D versucht so lange auf den Port zu verbinden, bis das Timeout (standardmässig 30 Sekunden) abgelaufen ist.

Beispiel
VarInt1 := TCP_IP_Connect(“192.168.1.1“; 4455)

Eine TCP/IP-Verbindung zum Host mit der IP-Adresse “192.168.1.1“ wird geöffnet. Der verwendete Port ist 4455.

TCP_IP_OpenServer

TCP_IP_OpenServer (Portnumber{; Timeout}) * -> Integer

Parameter	Typ	Beschreibung
Portnumber	Integer	Portnummer
Timeout	Integer	Zeit in Sekunden, in der sich ein Client verbinden kann
Funktionsergebnis:	Integer	TCP-Session-ID

Beschreibung
Mit 'TCP_IP_OpenServer' wird ein Server gestartet, der auf eine eingehende Verbindung auf dem angegebenen Port wartet. Optional kann die Wartezeit durch ein Timeout begrenzt werden, standardmässig wird unendlich lange gewartet. Zurückgegeben wird die TCP-Session-ID.

Beispiel
VarInt1:=TCPIP_OpenServer (4455)

Ein TCP/IP-Server wird auf Port 4455 geöffnet.

TCP_IP_CloseConnection

TCP_IP_CloseConnection (TCPSessionID) *

Parameter	Typ	Beschreibung
TCPSessionID	Integer	TCP-Session-ID

Beschreibung
Mit 'TCP_IP_CloseConnection' wird die übergebene Verbindung geschlossen, unabhängig davon ob Katana4D Server oder Client ist.

Beispiel
TCPIP_CloseConnection (VarInt1)

TCP_IP_Ping

TCP_IP_Ping (Hostname; Timeout) -> Integer

Parameter	Typ	Beschreibung
Hostname	String	Name oder IP des Hosts
Timeout	Integer	Timeout in Sekunden
Funktionsergebnis:	Integer	Status (1 oder 0)

Beschreibung
Mit 'TCP_IP_Ping' wird überprüft, ob der übergebene Host (z.B. “192.168.1.1“) antwortet oder nicht. Antwortet er innerhalb des übergebenen Timeouts, wird eine 1 zurückgegeben.

Beispiel
VarInt1:=TCP_IP_Ping (“192.168.1.1“; 1)

TCP_IP_Receive

TCP_IP_Receive (TCPSessionID) * -> Text

Parameter	Typ	Beschreibung
TCPSessionID	Integer	TCP-Session-ID
Funktionsergebnis:	Text	Buffer-Inhalt

Beschreibung
Mit 'TCPIP_Receive' wird der Buffer der übergebenen TCP/IP-Session geleert und als Text zurückgegeben.

Beispiel
VarText1:=TCPIP_Receive (VarInt1)

TCP_IP_Send

TCP_IP_Send (TCPSessionID; SendText) *

Parameter	Typ	Beschreibung

TCPSessionID	Integer	TCP-Session-ID
SendText	Text	Text, der gesendet werden soll

Beschreibung
Mit 'TCP_IP_Send' wird der Text 'SendText' an den Host der übergebenen TCP-Session gesendet.

Beispiel
TCP_IP_Send (VarInt1; VarText1)

2.3.5 ModBusTCP

ModBusTCP_Connect

ModBusTCP_Connect (Host-IP{; Timeout}{; Port}{; OutputOffset}) *

Parameter	Typ	Beschreibung
Host-IP	String	IP des Hosts
Timeout	Integer	Timeout in Sekunden
Port	Integer	Port des Hosts
OutputOffset	Integer	Offset für Adressierung der Ausgänge

Beschreibung
Mit 'ModBusTCP_Connect' kann mit dem übergebenen Host (nur IP erlaubt) über ModBus-TCP eine Verbindung aufgebaut werden. Über den optionalen Parameter 'Timeout' kann die Zeit, die höchstens für einen ModBusTCP-Befehl gebraucht werden darf (gilt auch für alle darauf folgenden Befehle, wie z.B. Lesen und Schreiben), angegeben werden. Standardwert ist eine Sekunde. Mit 'Port' kann der Standardwert von 502 geändert werden. Mit 'OutputOffset' kann die Adresse, bei der die Ausgänge beginnen, angegeben werden. Ist der erste Ausgang an Adresse 513, kann mit einem Offset von 512 erreicht werden, dass der erste Ausgang mit 1, der zweite mit 2 usw. angesprochen werden kann.
Tritt bei einem ModBusTCP-Befehl ein Fehler auf, wird direkt eine Fehlermeldung ausgegeben und das laufende Programm abgebrochen.

Beispiel
ModBusTCP_Connect (“192.168.1.1“)

ModBusTCP_Disconnect

ModBusTCP_Disconnect *

Parameter	Typ	Beschreibung

Dieser Befehl benötigt keine Parameter.

Beschreibung
'ModBusTCP_Disconnect' schliesst eine bestehende Verbindung.

ModBusTCP_ReadWord

ModBusTCP_ReadWord (Address) * -> Integer

Parameter	Typ	Beschreibung
Address	Integer	Adresse des Wortes
Funktionsergebnis:	Integer	Wort (16 Bit)

Beschreibung
Mit 'ModBusTCP_ReadWord' kann mit der übergebenen Adresse das gewünschte Wort auf dem ModBus-Server gelesen werden.
Die Funktion gibt anschliessend den Wert als 16bit-Integer (unsigned) zurück. Falls nötig können Sie die Konvertierungsfunktion 16BitINT_UnsignedTOSignedINT verwenden, um den Wert umzurechnen.

Beispiel
VarInt1 := ModBusTCP_ReadWord (1)

ModBusTCP_WriteWord

ModBusTCP_WriteWord (Address; Value) *

Parameter	Typ	Beschreibung
Adresse	Integer	Adresse des Wortes
Wert	Integer	Wort (16bit unsigned Integer)

Beschreibung
Mit 'ModBusTCP_WriteWord' kann ein Wort an der übergebenen Adresse auf einen gewünschten Wert gesetzt werden.
Es dürfen nur positive Zahlen übergeben werden (unsigned), sprich 0-65535. Wollen Sie serverseitig den Wert als signed interpretieren, können Sie den negativen Wert zuerst mit der Konvertierungsfunktion 16BitINT_SignedTOUnsignedINT umwandeln und erst dann mit ModBusTCP_WriteWord senden.

Beispiel
ModBusTCP_WriteWord(10; 4786)

ModBusTCP_ReadBit

ModBusTCP_ReadBit (Address; Position) * -> Integer

Parameter	Typ	Beschreibung
Address	Integer	Adresse des Wortes
Position	Integer	Bit-Position (0 - 15)
Funktionsergebnis:	Integer	Status (1 oder 0)

Beschreibung
Mit 'ModBusTCP_ReadBit' kann mit der übergebenen Adresse und der Position das gewünschte Bit des 16bit-Wortes auf dem ModBus-Server gelesen werden. Die Funktion gibt anschliessend den Status des Bits zurück.

Beispiel
VarInt1:=ModBusTCP_ReadBit (1; 2)

ModBusTCP_WriteBit

ModBusTCP_WriteBit (Address; Position; Status) *

Parameter	Typ	Beschreibung
Address	Integer	Adresse des Wortes
Position	Integer	Bit-Nr. (0-15)
Status	Integer	Status (1 oder 0)

Beschreibung
Mit 'ModBusTCP_WriteBit' kann mit der übergebenen Adresse und der Position das gewünschte Bit des 16bit-Wortes auf dem ModBus-Server auf 1 oder 0 gesetzt werden.
Achtung: Es wird nicht nur das einzelne Bit geschrieben. Zuerst wird der Wert eingelesen und erst anschliessend das komplette 16bit-Wort geschrieben.

Beispiel
ModBusTCP_WriteBit (1; 12; 1)

Beckhoff_Watchdog_Enable

Beckhoff_Watchdog_Enable {(Timeout)} *

Parameter	Typ	Beschreibung
Timeout	Integer	Timeout

Beschreibung
Aktiviert den Watchdog auf dem Beckhoff Buskoppler. Wird das Timeout nicht übergeben, wird der Standardwert von 1000 gesetzt.

Beckhoff_Watchdog_Disable

Beckhoff_Watchdog_Disable *

Beschreibung
Deaktiviert den Watchdog auf dem Buskoppler von Beckhoff.

2.3.6 Siemens S7 - Plugin

S7MPI_Connect

S7MPI_Connect (MPI-Address)

Parameter	Typ	Beschreibung
MPI-Address	Integer	MPI-Adresse

Beschreibung
'S7MPI_Connect' stellt eine Verbindung mit einer Siemens S7-3xx/4xx her.

Beispiel
S7MPI_Connect (2)

S7MPI_Disconnect

S7MPI_Disconnect

Beschreibung
Schliesst die momentan offene Verbindung. Dieser Befehl erwartet keine Parameter.

S7MPI_DBReadByte

S7MPI_DBReadByte (DBNr; DBBNr) -> Byte

Parameter	Typ	Beschreibung
DBNr	LongInteger	Datenbaustein Nummer
DBBNr	LongInteger	Byte-Nummer
Funktionsergebnis:	Byte	ausgelesener Wert

Beschreibung
'S7MPI_DBReadByte' liest vom Datenbaustein DBNr das Datenbyte DBBNr aus und gibt den Wert als Byte (signed; -127 bis 127) zurück.

Beispiel
Alert (String (S7MPI_DBReadByte (1; 0)))

S7MPI_DBReadWord

S7MPI_DBReadWord (DBNr; DBWNr) -> Integer

Parameter	Typ	Beschreibung
DBNr	LongInteger	Datenbaustein-Nummer
DBWNr	LongInteger	Byte-Nummer
Funktionsergebnis:	16bit Integer	ausgelesener Wert

Beschreibung
'S7MPI_DBReadWord' liest vom Datenbaustein DBNr zwei Datenbytes (ab DBWNr) aus und gibt den Wert als 16bit Integer (signed; -32767 bis 32767) zurück.

Beispiel
Alert (String (S7MPI_DBReadWord (1; 0)))

S7MPI_DBReadDWord

S7MPI_DBReadDWord (DBNr; DBWNr) -> 32bit Integer

Parameter	Typ	Beschreibung
DBNr	LongInteger	Datenbaustein-Nummer
DBWNr	LongInteger	Byte-Nummer
Funktionsergebnis:	32bit Integer	ausgelesener Wert

Beschreibung
'S7MPI_DBReadWord' liest vom Datenbaustein DBNr vier Datenbytes (ab DBDNr) aus und gibt den Wert als 32bit Integer (signed; -2’147’483’647 bis 2’147’483’647) zurück.

Beispiel
Alert (String (S7MPI_DBReadDWord (1; 0)))

S7MPI_DBReadString

S7MPI_DBReadString (DBNr; DBBNr) -> Char

Parameter	Typ	Beschreibung
DBNr	LongInteger	Datenbaustein-Nummer
DBBNr	LongInteger	Byte-Nummer
Funktionsergebnis:	Char	ausgelesener Wert

Beschreibung
'S7MPI_DBReadString' liest aus dem Datenbaustein DBNr das Datenbyte DBBNr aus und gibt den Wert als Char zurück.

Beispiel
Alert (S7MPI_DBReadString (1; 0))

Das Beispiel liest aus dem Datenbaustein DB01 das Datenbyte 0 (Adresse in Step7: DB1.DBB 0) und zeigt den Wert in einer Alert-Meldung.

S7MPI_DBWriteByte

S7MPI_DBWriteByte (DBNr; DBBNr; Value)

Parameter	Typ	Beschreibung
DBNr	LongInteger	Datenbaustein-Nummer
DBBNr	LongInteger	Byte-Nummer
Value	Integer	Wert

Beschreibung
Mit den ersten zwei Parametern werden die Datenbaustein-Nummer und die Datenbyte-Nummer angegeben. Als dritter Parameter wird eine Zahl zwischen -127 und 127 übergeben, die in das Datenbyte geschrieben werden soll.

Beispiel
S7MPI_DBWriteByte (1; 2; -96)

Speichern der Zahl -96 in Datenbaustein DB1.DBB Byte 2.

S7MPI_DBWriteWord

S7MPI_DBWriteWord (DBNr; DBWNr; Value)

Parameter	Typ	Beschreibung
DBNr	LongInteger	Datenbaustein-Nummer
DBWNr	LongInteger	Byte-Nummer
Value	Integer	Wert

Beschreibung
Mit den ersten zwei Parametern werden die Datenbaustein-Nummer und die Byte-Nummer angegeben. Als dritter Parameter wird eine Zahl zwischen -32767 und 32767 übergeben, die in das Datenwort geschrieben werden soll.

Beispiel
S7MPI_DBWriteWord(1; 2; -32000)

Speichern der Zahl -32'000 in Datenbaustein DB1.DBW Byte 2.

S7MPI_DBWriteDWord

S7MPI_DBWriteDWord (DBNr; DBDNr; Value)

Parameter	Typ	Beschreibung
DBNr	LongInteger	Datenbaustein-Nummer
DBDNr	LongInteger	Byte-Nummer
Value	Integer	Wert

Beschreibung
Mit den ersten zwei Parametern werden die Datenbaustein-Nummer und die Byte-Nummer angegeben. Als dritter Parameter wir eine Zahl zwischen -2'147'483'647 und 2'147'483'647 übergeben, die in das Datendoppelwort geschrieben werden soll.

Beispiel
S7MPI_DBWriteDWord (1; 2; -350000)

Speichern der Zahl -350'000 in Datenbaustein DB1.DBD Byte 2.

S7MPI_DBWriteString

S7MPI_DBWriteString (DBNr; DBBNr; Value)

Parameter	Typ	Beschreibung
DBNr	LongInteger	Datenbaustein-Nummer
DBBNr	LongInteger	Byte-Nummer
Value	String	Wert

Beschreibung
Mit 'S7MPI_DBWriteString' kann ein String in einen Datenbaustein geschrieben werden. Jedes einzelne Zeichen wird in einem separaten Datenbyte gespeichert.
Der erste Parameter ist die Nummer des Datenbausteins, der zweite gibt an, ab welchem Datenbyte geschrieben werden soll und der dritte ist der Wert, der geschrieben werden soll. Wird ein String mit 5 Zeichen übergeben, werden ab DBBNr fünf Bytes geschrieben.

Beispiel
S7MPI_DBWriteString (1; 10; “hallo“)

Das Beispiel zeigt wie die Bytes 10 bis 14 vom Datenbaustein DB1.DBB mit den Zeichen “h“, “a“, “l“, “l“, und “o“ beschrieben werden.

S7MPI_ReadDigitalInputByte

S7MPI_ReadDigitalInputByte (ByteNr) -> Integer

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
Funktionsergebnis:	Integer	ausgelesener Wert

Beschreibung
Es werden alle 8 Bits des digitalen Eingang-Bytes ByteNr gelesen und als Integer ausgegeben (0-255).

Beispiel
Alert (String (S7MPI_ReadDigitalInputByte(124)))

Das Beispiel liest Eingangs-Byte 124 und gibt die Zahl als Alert-Meldung aus.

S7MPI_ReadDigitalOutputByte

S7MPI_ReadDigitalOutputByte (ByteNr) -> Integer

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Datenbaustein-Nummer
Funktionsergebnis:	Integer	ausgelesener Wert

Beschreibung
Es werden alle 8 Bits des digitalen Ausgangs-Bytes 'ByteNr' gelesen und als Integer zurückgegeben.

Beispiel
Alert (String (S7MPI_ReadDigitalOutputByte (1)))

Liest Output-Byte 1 und gibt die Zahl als Alert-Meldung aus.

S7MPI_ReadDigitalInputBit

S7MPI_ReadDigitalInputBit (ByteNr; BitNr) -> Boolean

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
BitNr	LongInteger	Bit-Nummer
Funktionsergebnis:	Boolean	Status des ausgelesenen Bits

Beschreibung
Gibt den Wert eines einzigen Bits (BitNr) von einem digitalen Eingangs-Byte (ByteNr) als Boolean zurück.

Beispiel
VarBool1 := S7MPI_ReadDigitalInputBit(124; 0)

Liest den Status von Bit 0 des Eingangs-Bytes 124 und speichert diesen in der Variable 'VarBool1'.

S7MPI_ReadDigitalOutputBit

S7MPI_ReadDigitalOutputBit (ByteNr; BitNr) -> Boolean

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
BitNr	LongInteger	Bit-Nummer
Funktionsergebnis:	Boolean	ausgelesener Wert

Beschreibung
'S7MPI_ReadDigitalOutputBit' gibt den Wert von einem Bit ('BitNr') aus einem digitalen Ausgangs-Byte ('ByteNr') als Boolean zurück.

Beispiel
S7MPI_ReadDigitalInputBit(126;7)

Bit 7 von Ausgang 126 wird gelesen.

S7MPI_WriteDigitalOutputByte

S7MPI_WriteDigitalOutputByte (ByteNr; Value)

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
Value	Integer	Wert

Beschreibung
Mit 'S7MPI_WriteDigitalOutputByte' können digitale Ausgänge gesetzt werden. Es kann nur ein ganzes Byte geschrieben werden. Einzelne Ausgangs-Bits zu setzen ist nicht möglich.

Beispiel
S7MPI_WriteDigitalOutputByte (1; 0)
S7MPI_WriteDigitalOutputByte (1; 255)

Alle Ausgang-Bits von Ausgang-Byte 1 werden zuerst auf 0 und dann auf 1 gesetzt.

S7MPI_ReadFlagBit

S7MPI_ReadFlagBit (ByteNr; BitNr) -> Boolean

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
BitNr	LongInteger	Bit-Nummer
Funktionsergebnis:	Boolean	Status des ausgelesenen Flag

Beschreibung
'S7MPI_ReadFlagBit' liest den Status vom Merker 'ByteNr' an der Stelle 'BitNr' und gibt den Status des Bits als Boolean zurück.

Beispiel
VarBool1 := S7MPI_ReadFlagBit(4; 7).

Liest den Status von Bit 7 des vierten Merkers und speichert diesen in der Variable 'VarBool1'.

S7MPI_ReadFlagByte

S7MPI_ReadFlagByte(ByteNr) -> Integer

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
Funktionsergebnis:	Integer	ausgelesener Wert

Beschreibung
'S7MPI_ReadFlagByte' liest ein Byte vom Merker 'ByteNr'.

Beispiel
Alert (String (S7MPI_ReadFlagByte(1)))

Gibt den Wert (Zahl) des ersten Flag als Alert-Meldung aus.

S7MPI_ReadFlagWord

S7MPI_ReadFlagWord (ByteNr) -> Integer

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
Funktionsergebnis:	Integer	ausgelesener Wert

Beschreibung
'S7MPI_ReadFlagWord' liest ein Wort vom Merker 'ByteNr'.

Beispiel
Alert (String (S7MPI_ReadFlagWord(1)))

Gibt den Wert (Wort) des ersten Flag als Alert-Meldung aus.

S7MPI_ReadFlagDWord

S7MPI_ReadFlagDWord (ByteNr) -> Integer

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
Funktionsergebnis	Integer	ausgelesener Wert

Beschreibung
'S7MPI_ReadFlagDWord' liest eine Zahl vom Merker-Doppelwort 'ByteNr'.

Beispiel
Alert (String (S7MPI_ReadFlagDWord(1)))

Gibt den Wert (Doppelwort) des ersten Flag als Alert-Meldung aus.

S7MPI_ReadFlagString

S7MPI_ReadFlagString (ByteNr) -> Char

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
Funktionsergebnis:	Char	ausgelesener Wert

Beschreibung
'S7MPI_ReadFlagString' liest ein Zeichen vom Merker 'ByteNr'.

Beispiel
Alert (S7MPI_ReadFlagString(1))

Gibt den Wert (Zeichen) des ersten Flag als Alert-Meldung aus.

S7MPI_WriteFlagBit

S7MPI_WriteFlagBit (ByteNr; BitNr; Value)

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
BitNr	LongInteger	Bit-Nummer
Value	LongInteger	Wert (0 oder 1)

Beschreibung
'S7MPI_WriteFlagBit' setzt im Merker 'ByteNr' das Bit 'BitNr'. Das Bit wird nur gesetzt, wenn 1 übergeben wird..

Beispiel
S7MPI_WriteFlagBit(4; 7; 1)

Setzt Bit 7 des vierten Flag auf 1.

S7MPI_WriteFlagByte

S7MPI_WriteFlagByte (ByteNr; Value)

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
Value	Integer	Wert

Beschreibung
Mit 'S7MPI_WriteFlagByte' kann eine Zahl zwischen -127 und 127 in ein Merker-Byte geschrieben werden.

Beispiel
S7MPI_WriteFlagByte(4;15)

Schreibt die Zahl 15 in Merker-Byte 4.

S7MPI_WriteFlagWord

S7MPI_WriteFlagWord (ByteNr; Value)

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
Value	Integer	Wert

Beschreibung
Mit 'S7MPI_WriteFlagWord' kann eine Zahl (-32767 bis +32767) in ein Merker-Wort schreiben.

Beispiel
S7MPI_WriteFlagWord(3;15)

Schreibt die Zahl 800 in Merker-Wort 3.

S7MPI_WriteFlagDWord

S7MPI_WriteFlagDWord (ByteNr; Value)

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
Value	Integer	Wert

Beschreibung
Mit 'S7MPI_WriteFlagDWord' kann eine Zahl zwischen -2’147’483’647 und 2'147'483'647 in ein Merker-Doppelwort geschrieben werden.

Beispiel
S7MPI_WriteFlagDWord (1; 48000)

Schreibt die Zahl 48'000 in Merker-Doppelwort 1.

S7MPI_WriteFlagString

S7MPI_WriteFlagString(ByteNr; Value)

Parameter	Typ	Beschreibung
ByteNr	LongInteger	Byte-Nummer
Value	String	Wert

Beschreibung
Es werden ab Merker-Byte 'ByteNr' so viele Bytes geschrieben wie 'Value' Zeichen enthält.

Beispiel
S7MPI_WriteFlagString(1; ”test”)

Schreibt in Merker-Byte 1 bis 4 das Wort “test“ (wie bei S7MPI_ DBWriteString, immer ein Zeichen pro Byte).

2.3.7 Controlbox

CB_SetDigitalOutput

CB_SetDigitalOutput (OutputNr; Value) *

Parameter	Typ	Beschreibung
OutputNr	Integer	Output-Nummer
Value	Integer	Zustand (0 oder 1)

Beschreibung
'CB_SetDigitalOutput' setzt einen digitalen Ausgang auf der Controlbox des Katanas.

Beschriftung	Outputnummer
'A'	1
'B'	2

Beispiel
CB_SetDigitalOutput (2; 1)

Setzt den Ausgang 2 auf 1.

CB_ReadDigitalInput

CB_ReadDigitalInput (InputNr) * -> Integer

Parameter	Typ	Beschreibung
InputNr	Integer	Input-Nummer
Funktionsergebnis	Integer	Zustand (0 oder 1)

Beschreibung
'CB_ReadDigitalInput' liest einen digitalen Eingang ein.

Beschriftung	Inputnummer
'A'	1
'B'	2
'C'	3
'D'	4
'E'	5
'F'	6

Beispiel
VarInt1 := CB_ReadDigitalInput (1)

Liest den Eingang 1 und speichert den Wert in der Variable 'VarInt1' ab.

CB_SetLED

CB_SetLED (“State“) *

Parameter	Typ	Beschreibung
State	Integer	Zustand der LED

Beschreibung
'CB_SetLED' setzt den Zustand der LED auf der Controlbox.

Zustände:	0	LED ausgeschaltet
	1	LED leuchtet grün
	2	LED leuchtet rot

Beispiel
CB_SetLED(0)

Schaltet die LED aus.

2.4 Programmablauf-Steuerung

Break

Break *

Beschreibung
Katana4D berücksichtigt beim Programmablauf jeweils auch die nächste Zeile. Folgt auf einen Aktionsbefehl eine weitere Bewegung, wird der entsprechende Fahrbefehl schon während der aktuellen Bewegung an die Controlbox geschickt. Auf diese Weise kann die Wartezeit zwischen zwei Befehlen eliminiert werden. Dies kann geändert werden, indem nach einem Bewegungsbefehl 'Break' verwendet wird. Dadurch wartet das Programm, bis die aktuelle Bewegung zu Ende ist, bevor die nächste ausgeführt wird.
'Break' hat keinen Einfluss auf das Programm, wenn es nach einem Befehl ausgeführt wird, der kein Aktionsbefehl ist.

If Then Else

If Then Else *

```
If (Bedingung)
   `Benutzer-Code
Else
   `Benutzer-Code
End If
```

Parameter	Typ	Beschreibung
Bedingung	Boolean	Bedingung

Beschreibung
Wenn die Bedingung zutrifft, werden alle Befehle zwischen dem “If“ und dem “Else“ ausgeführt. Ansonsten diejenigen zwischen “Else“ und “End If“.

Beispiel

```
Confirm (“Ja?“)
If (OK = 1)
   Alert (“Ok“)
Else
   Alert (“Abbrechen“)
End If
```

Es wird ein Bestätigungsfenster (“Ja?“) angezeigt. Wenn mit OK bestätigt wird, erscheint die Meldung “OK“, sonst “Abbrechen“.

Repeat Until

Repeat Until *

```
Repeat
   `Benutzer-Code
Until (Bedingung)
```

Parameter	Typ	Beschreibung
Bedingung	Boolean	Bedingung

Beschreibung
Die Befehle zwischen “Repeat“ und “Until“ werden solange ausgeführt, bis die Bedingung zutrifft.

Beispiel

```
i := 0
Repeat
   i := i + 1
   Alert (String (i))
Until (i = 5)
```

Die Schleife wird fünfmal ausgeführt. Bei jedem Durchgang wird eine Meldung mit dem Wert der Zählervariable ausgegeben.

Wait

Wait (Millisekunden) *

Parameter	Typ	Beschreibung
Millisekunden	LongInteger	Wartezeit in Millisekunden

Beschreibung
'Wait' erlaubt es, die Programmfortsetzung an beliebiger Stelle für eine in Milli-sekunden angegebene Zeitdauer anzuhalten.

RunUserMethod

RunUserMethod (“UserMethodName“) *

Parameter	Typ	Beschreibung
UserMethodName	String	Name der Benutzermethode

Beschreibung
Führt den Code aus, der als Benutzermethode gespeichert wurde. Der Name der Benutzermethode muss als String übergeben werden. Benutzermethoden können beliebig viele Zeilen Code enthalten. Ebenso kann in einer Benutzermethode auf andere Benutzermethoden zugegriffen werden.

Hinweis
Eine Benutzermethode kann auch ausgeführt werden, indem man den Namen der Methode direkt in das Script schreibt.

Beispiele
RunUserMethod(“MeineErsteMethode”)
MeineErsteMethode

Beide Varianten führen die Benutzermethode aus.

GetErrorMessage

GetErrorMessage -> String

Parameter	Typ	Beschreibung
Funktionsergebnis:	String	Fehlermeldung

Beschreibung
'GetErrorMessage' liefert eine Fehlermeldung zurück. Nachdem ein Fehler aufgetreten ist, kann man die Meldung mit dieser Funktion abrufen.

Hinweis
Die ereignisabhängige Benutzermethode 'OnError' wird bei einem Fehler ausgeführt. In dieser Methode kann 'GetErrorMessage' die Meldung des Fehlers abfragen.

Login

Login ({User{; Password}})

Parameter	Typ	Beschreibung
User	String	Benutzername
Password	String	Passwort des Benutzers

Beschreibung
Wird kein Parameter übergeben, öffnet sich das Login-Fenster, in dem der Benutzername und das Passwort eingegeben werden kann.
Falls nur der Benutzername übergeben wird, erscheint dasselbe Fenster. Der Benutzername ist aber bereits eingetragen.
Bei einer Übergabe von Namen und Passwort wird der entsprechende Benutzer direkt angemeldet, sofern die Eingaben korrekt sind. Andernfalls erscheint eine Fehlermeldung.

Logout

Logout

Beschreibung
Abmelden des Benutzers. Nach der Abmeldung wird der Startbildschirm angezeigt.

2.4.1 Input/Output

Alert

Alert (“Text“)

Parameter	Typ	Beschreibung
Text	String	Meldungstext

Beschreibung
Das Programm wird unterbrochen und eine Warnmeldung in einem neuen Fenster ausgegeben. Nach einem Klick auf “OK“ wird das Programm fortgeführt.

Confirm

Confirm (“Text“{; “OKTitle“{; “CancelTitle“}})

Parameter	Typ	Beschreibung
Text	String	Fragetext
OKTitle	String	Text für den OK-Button
CancelTitle	String	Text für den Abbrechen-Button

Beschreibung
Gibt eine Frage aus, auf die mit OK (Standard) oder Abbrechen (Standard) reagiert werden kann. Das Ergebnis wird in die globale Variable “OK“ geschrieben:

OK-Button -> OK = 1
Abbrechen-Button -> OK = 0

Beispiel

```
Confirm (“Ja?“)
If (OK = 1)
   Alert (“Ok“)
Else
   Alert (“Abbrechen“)
End If
```

Das Ergebnis von 'Confirm' wird als Alert-Meldung ausgegeben.

Request

Request (“Message“{; “DefaultResponse“{; “OKTitle“{; “CancelTitle“}}}) -> String

Parameter	Typ	Beschreibung
Message	String	Text
DefaultResponse	String	Vorgabewert
OKTitle	String	Text für OK-Button
CancelTitle	String	Text für Abbrechen-Button
Funktionsergebnis	String	vom Benutzer eingegebener Wert

Beschreibung
'Request' öffnet ein Fenster mit einem Eingabefeld. Mit dem Parameter 'DefaultResponse' kann ein Vorgabewert definiert werden. Der Dialog kann mit OK (Standard) oder Abbrechen (Standard) bestätigt werden.

Beispiel
VarStr1 := Request (“Rasterpunkt?“; “1“)

PostMessage

PostMessage (“Message“)

Parameter	Typ	Beschreibung
Text	String	Meldungstext

Beschreibung
'PostMessage' zeigt einen Meldungstext in einem eigenen Fenster an. Das Fenster verschwindet allerdings wieder, wenn ein neues Formular geöffnet wird. Im Gegensatz zu 'Alert' wird das Programm während der Meldung nicht unterbrochen.

OpenWindow

OpenWindow (Width; Height; “WindowTitle“)

Parameter	Typ	Beschreibung
Width	Integer	Breite in Pixel
Height	Integer	Höhe in Pixel
WindowTitle	String	Titel des Fensters

Beschreibung
'OpenWindow' öffnet ein neues Fenster, das man für Ausgaben verwenden kann.

2.4.2 Program

RunProgram

RunProgram (Nr; Repetitions)

Parameter	Typ	Beschreibung
Nr	Integer	Programm-Nummer
Repetitions	Integer	Anzahl Wiederholungen

Beschreibung
'RunProgram' startet das Programm mit der übergebenen Nummer.

RunApplication

RunApplication (“Name“)

Parameter	Typ	Beschreibung
Nr	String	Anwendungsname

Beschreibung
'RunApplication' startet die übergebene Anwendung.

ExitProgram

ExitProgram (Nr)

Parameter	Typ	Beschreibung
Nr	Integer	Programm-Nummer

Beschreibung
'ExitProgram' stoppt ein laufendes Programm. Achtung: Ein Programm kann sich nicht selber mit 'ExitProgram' beenden.

ExitApplication

ExitApplication (“Name“)

Parameter	Typ	Beschreibung
Nr	String	Anwendungsname

Beschreibung
'ExitApplication' beendet die übergebene Anwendung.

PauseProgram

PauseProgram (Nr{; Timeout}) -> Integer

Parameter	Typ	Beschreibung
Nr	Integer	Programm-Nummer
Timeout	Integer	Wartezeit in Millisekunden
Funktionsergebnis:	Integer	Status

Beschreibung
Mit 'PauseProgram' kann ein laufendes Programm angehalten werden. Das angehaltene Programm kann mit 'ResumeProgram' fortgeführt werden. Ein Programm wird immer zwischen zwei Befehlen angehalten und nicht während der Ausführung eines Befehls. Das bedeutet, dass es zwischen dem Ausführen von 'PauseProgram' und dem effektiven Anhalten zu einer zeitlichen Verschiebung kommen kann.
Wird nur der erste Parameter übergeben, wird das entsprechende Programm angehalten und 1 zurückgegeben. Werden aber beide Parameter übergeben, wird so lange gewartet, bis das Timeout abgelaufen ist oder sich das Programm angehalten hat. Achtung: Das Programm wird auch nach dem Ablaufen des Timeouts angehalten. 'Timeout' sollte also genügend gross gewählt werden. Hat sich das Programm innerhalb der gewünschten Zeit angehalten, wird 1 zurückgegeben. Falls das Timeout abgelaufen ist oder das anzuhaltende Programm nicht existiert, wird 0 zurückgegeben. Wenn für 'Timeout' 0 übergeben wird, wird endlos gewartet.

Beispiel
VarInt1:=PauseProgram(1;10000)

Programm 1 muss innert 10 Sekunden angehalten werden. Wenn dies nicht erfolgen kann, wird 0 zurückgegeben.

ResumeProgram

ResumeProgram (Nr) -> Integer

Parameter	Typ	Beschreibung
Nr	Integer	Programm-Nummer
Funktionsergebnis:	Integer	Status

Beschreibung
'ResumeProgram' startet das übergebene Programm und 1 wird zurückgeben. Existiert das Programm nicht wird 0 zurückgegeben.

ProgramState

ProgramState (Nr) -> Integer

Parameter	Typ	Beschreibung
Nr	Integer	Programm-Nummer
Funktionsergebnis:	Integer	Programm-Status

Beschreibung
'ProgramState' gibt den aktuellen Status des übergebenen Programms zurück. Ein Programm kann fünf verschiedene Zustände haben:

0	Programm wird nicht ausgeführt
1	Programm wird ausgeführt
2	Programm ist angehalten
3	Programm schläft
4	Programm ist unterbrochen, da ein Fenster angezeigt wird, das vom Benutzer bestätigt werden muss

2.5 Mathematische Funktionen

Abs

Abs (Value) * -> Numerisch

Parameter	Typ	Beschreibung
Value	Numerisch	Umzuwandelnder Wert
Funktionsergebnis:	Numerisch	Absoluter Wert von 'Value'

Beschreibung
'Abs' gibt den absoluten Wert von 'Value' zurück. Ist der Wert negativ, wird der positive Wert zurückgegeben. Ist er positiv, wird der Wert unverändert zurückgegeben.

Arctan

Arctan (Angle) -> Numerisch

Parameter	Typ	Beschreibung
Angle	Numerisch	Wert, für den der Winkel im Bogenmass berechnet werden soll.
Funktionsergebnis:	Numerisch	Winkel im Bogenmass

Beschreibung
'Arctan' gibt den Arcustangens (Winkel im Bogenmass) von 'Angle' zurück.

Cos

Cos (Angle) -> Numerisch

Parameter	Typ	Beschreibung
Angle	Numerisch	Winkel im Bogenmass
Funktionsergebnis:	Numerisch	Cosinus des Wertes

Beschreibung
'Cos' gibt den Cosinus des im Bogenmass angegebenen Winkels zurück.

Hinweis
Winkel in Grad müssen zuerst ins Bogenmass umgerechnet werden. Dazu werden sie mit der Konstante 'Degree' (0.01745...) multipliziert.

Dec

Dec (Value) -> Numerisch

Parameter	Typ	Beschreibung
Value	Numerisch	Umzuwandelnder Wert
Funktionsergebnis:	Numerisch	Dezimalstellen von Wert

Beschreibung
'Dec' gibt die Nachkommastellen von 'Value' zurück. Das Ergebnis ist immer zwischen 0 und 1.

Exp

Exp (Value) -> Numerisch

Parameter	Typ	Beschreibung
Value	Numerisch	Positive Zahl ungleich Null
Funktionsergebnis:	Numerisch	Natürliche Exponentialfunktion von 'Value'

Beschreibung
'Exp' gibt den Exponent von 'Value' zurück. 'Exp' ist die Umkehrfunktion von 'Log'.

Int

Int (Value) * ->Numerisch

Parameter	Typ	Beschreibung
Value	Numerisch	Umzuwandelnder Wert
Funktionsergebnis:	Integer	Integer von 'Value'

Beschreibung
'Int' gibt den abgerundeten, ganzzahligen Wert von 'Value' zurück.

Log

Log (Value) -> Numerisch

Parameter	Typ	Beschreibung
Value	Numerisch	Positive Zahl ungleich Null
Funktionsergebnis:	Numerisch	Logarithmus von 'Value'

Beschreibung
'Log' gibt den natürlichen Logarithmus von 'Value' zurück. 'Log' ist die Umkehrfunktion von 'Exp'.

Mod

Mod (Value; Divisor) -> Numerisch

Parameter	Typ	Beschreibung
Value	Numerisch	Zu dividierende Zahl
Divisor	Numerisch	Divisor
Funktionsergebnis:	Numerisch	Gibt den ganzzahligen Rest der Division zurück

Beschreibung
'Mod' gibt als Ergebnis den ganzzahligen Rest aus der Division 'Value' durch 'Divisor' zurück. Statt dieser Funktion kann auch der Operator '%' verwendet werden. Er liefert jedoch nur für Integer oder LongInteger gültige Ergebnisse.

Hinweis
Ist 'Value' vom Typ Real, wird 'Value' zuerst gerundet.

Random

Random -> Numerisch

Parameter	Typ	Beschreibung
Funktionsergebnis:	Integer	Zufallszahl

Beschreibung
'Random' gibt einen Integer zwischen 0 und 32767 (einschliesslich) zurück.

Round

Round (Value; Places) * -> Numerisch

Parameter	Typ	Beschreibung
Value	Numerisch	zu rundender Wert
Places	Integer	Anzahl Dezimalstellen
Funktionsergebnis:	Numerisch	gerundeter Wert

Beschreibung
'Round' gibt den, bis auf die in 'Places' angegebene Anzahl Stellen, gerundeten Wert zurück. Stellen zwischen 5 und 9 werden aufgerundet, solche zwischen 0 und 4 abgerundet.

Sin

Sin (Angle) -> Numerisch

Parameter	Typ	Beschreibung
Angle	Numerisch	Winkel im Bogenmass
Funktionsergebnis:	Numerisch	Sinus von 'Angle'

Beschreibung
'Sin' gibt den Sinus von 'Value' (Bogenmass) zurück.

Hinweis
Winkel in Grad müssen zuerst ins Bogenmass umgerechnet werden. Dazu werden sie mit der Konstante 'Degree' (0.01745...) multipliziert.

Square root

Square root (Value) -> Numerisch

Parameter	Typ	Beschreibung
Value	Numerisch	Wert, dessen Quadratwurzel berechnet werden soll
Funktionsergebnis:	Numerisch	Quadratwurzel von 'Value'

Beschreibung
'Square root' gibt die Quadratwurzel von 'Value' zurück.

Tan

Tan (Angle) -> Numerisch

Parameter	Typ	Beschreibung
Angle	Numerisch	Winkel im Bogenmass
Funktionsergebnis:	Numerisch	Tangens von 'Angle'

Beschreibung
'Tan' gibt den Tangens von 'Value' (Bogenmass) zurück.

Hinweis
Winkel in Grad müssen zuerst ins Bogenmass umgerechnet werden. Dazu werden sie mit der Konstante 'Degree' (0.01745...) multipliziert.

Trunc

Trunc (Value; Places) -> Numerisch

Parameter	Typ	Beschreibung
Value	Numerisch	Abzuschneidender Wert
Places	Integer	Anzahl Dezimalstellen
Funktionsergebnis:	Numerisch	abgeschnittener Wert

Beschreibung
'Trunc' gibt den, bis auf die in 'Places' angegebene Anzahl Stellen, “abgeschnittenen“ Wert von 'Value' zurück. Beim Abschneiden wird immer auf die niedrigere Zahl gerundet.

2.6 Datum & Zeit

Add to date

Add to date (Date; Year; Month; Day) -> Datum

Parameter	Typ	Beschreibung
Date	Datum	Datum, zu dem Tage, Monate und Jahre hinzugefügt werden sollen
Year	Numerisch	Anzahl der hinzuzufügenden Jahre
Month	Numerisch	Anzahl der hinzuzufügenden Monate
Day	Numerisch	Anzahl der hinzuzufügenden Tage
Funktionsergebnis:	Datum	Resultierendes Datum

Beschreibung
'Add to date' fügt dem in 'Date' übergebenen Datum 'Year', 'Month' und 'Day' hinzu und gibt das Ergebnis zurück.

Current date

Current date -> Datum

Parameter	Typ	Beschreibung
Funktionsergebnis:	Datum	Aktuelles Datum

Beschreibung
'Current date' gibt das aktuelle Datum zurück. Es entspricht dem Datum des Rechners.

Current time

Current time -> Zeit

Parameter	Typ	Beschreibung
Funktionsergebnis	Zeit	Aktuelle Zeit

Beschreibung
'Current time' gibt die aktuelle Systemzeit des Rechners zurück.

Date

Date (Text) -> Datum

Parameter	Typ	Beschreibung
Text	String	Text, der in ein Datum umgewandelt werden soll
Funktionsergebnis:	Datum	Datum

Beschreibung
'Date' gibt ein Ergebnis vom Typ Datum aus der Umwandlung des Textes Text zurück.
'Text' muss folgendes Format haben: TT.MM.JJ

Beispiel
varDate := Date(“01.01.09“)

Day of

Day of (Date) -> Numerisch

Parameter	Typ	Beschreibung
Date	Datum	Datum, dessen Tag ermittelt werden soll
Funktionsergebnis:	Numerisch	Tag innerhalb eines Monats

Beschreibung
'Day of' gibt einen Integer zwischen 1 und 31, der dem Tag von 'Date' entspricht, zurück.

Day number

Day number (Date) -> Numerisch

Parameter	Typ	Beschreibung
Date	Datum	Datum, dessen Wochentag ermittelt werden soll
Funktionsergebnis:	Numerisch	Zahl für den Wochentag, auf den 'Date' fällt

Beschreibung
'Day number' gibt einen Integer zwischen 1 und 7, der dem Wochentag von 'Date' entspricht, zurück.
1 = Sonntag, ..., 7 = Samstag

Milliseconds

Milliseconds -> LongInteger

Parameter	Typ	Beschreibung
Funktionsergebnis:	LongInteger	Anzahl Millisekunden

Beschreibung
'Milliseconds' gibt die Anzahl Millisekunden seit dem Start des Rechners zurück.

Month of

Month of (Date) -> Numerisch

Parameter	Typ	Beschreibung
Date	Datum	Datum, dessen Monat ermittelt werden soll
Funktionsergebnis:	Numerisch	Zahl, die den Monat für ein Datum angibt.

Beschreibung
'Month of' gibt einen Integer zwischen 1 und 12, der dem Monat von 'Date' entspricht, zurück.

Time

Time (Text) -> Zeit

Parameter	Typ	Beschreibung
Text	String	In Zeit umzuwandelnder Text
Funktionsergebnis:	Zeit	Zeit

Beschreibung
'Time' gibt ein Ergebnis vom Typ Zeit aus der Umwandlung von 'Text' zurück.

Beispiel
varTime := Time(“12:27:08“)

Time string

Time string (Time) -> String

Parameter	Typ	Beschreibung
Time	Numerisch	Zeit, die in einen Text umgewandelt werden soll
Funktionsergebnis:	String	Zeit als String

Beschreibung
'Time string' gibt den alphanumerischen Wert zurück, der 'Time' entspricht.

Year of

Year of (Date) ->Numerisch

Parameter	Typ	Beschreibung
Date	Datum	Datum, dessen Jahr ermittelt werden soll
Funktionsergebnis:	Numerisch	Zahl, die dem Jahr von 'Date' entspricht

Beschreibung
'Year of' gibt einen Integer, der dem Jahr von 'Date' entspricht, zurück.

2.7 String

Ascii

Ascii (Character) -> Numerisch

Parameter	Typ	Beschreibung
Character	String	Text, von dessen erstem Zeichen der ASCII-Code ermittelt werden soll
Funktionsergebnis:	Numerisch	ASCII-Code für das erste Zeichen

Beschreibung
'Ascii' gibt den ASCII-Code des ersten Zeichens von 'Character' zurück.

Change string

Change string (Source; Destination; Position) -> String

Parameter	Typ	Beschreibung
Source	String	Text, in dem Teile ersetzt werden sollen
Destination	String	Neue Zeichen
Position	Numerisch	Beginn der Änderung
Funktionsergebnis:	String	Geänderter Text

Beschreibung
'Change string' ersetzt eine Zeichengruppe in 'Source' durch 'Destination' und gibt diesen Text zurück. 'Change string' gibt immer einen String mit derselben Länge wie 'Source' zurück.
Wird für 'Destination' ein leerer String (“”) übergeben, bleibt 'Source' unverändert. Ist die Länge von 'Destination' kleiner als 1 oder länger als die von 'Source', bleibt 'Source' ebenfalls unverändert.
Im Gegensatz zu 'Insert' fügt 'Change string' keine Zeichen ein, sondern überschreibt sie nur.

Char

Char (Nr) -> String

Parameter	Typ	Beschreibung
Nr	Numerisch	Nummer des ASCII Zeichen (0 bis 255)
Funktionsergebnis:	String	Zeichen des ASCII-Code

Beschreibung
'Char' gibt ein Zeichen zurück, dessen ASCII-Code 'Nr' entspricht.

Delete string

Delete string (Source; Position; Length) -> String

Parameter	Typ	Beschreibung
Source	String	Text, in dem gelöscht werden soll
Position	Numerisch	Startposition des Löschens
Length	Numerisch	Anzahl der zu löschenden Zeichen
Funktionsergebnis	String	Geänderter Text

Beschreibung
'Delete string' löscht die durch 'Length' bestimmte Anzahl Zeichen ab 'Position' und gibt diesen Text zurück. 'Source' bleibt unverändert.

Insert string

Insert string (Source; Destination; Position) * -> String

Parameter	Typ	Beschreibung
Source	String	Text, in den eingefügt werden soll
Destination	String	Einzufügender Text
Position	Numerisch	Startposition des Einfügens
Funktionsergebnis:	String	Geänderter Text

Beschreibung
'Insert string' fügt 'Destination' in 'Source' ein und gibt diesen Text zurück. 'Position' markiert den Beginn des Einfügens. 'Source' bleibt unverändert. Die Zeichen hinter 'Position' werden an das Ende des Textes verschoben.

Length

Length * (Text) -> Numerisch

Parameter	Typ	Beschreibung
Text	String	Text, dessen Länge berechnet werden soll
Funktionsergebnis:	Numerisch	Länge von 'Text'

Beschreibung
'Length' gibt die Anzahl Zeichen von 'Text' zurück.

Lowercase

Lowercase (Text) -> String

Parameter	Typ	Beschreibung
Text	Text	In Kleinbuchstaben umzuwandelnder Text
Funktionsergebnis:	String	Text in Kleinbuchstaben

Beschreibung
'Lowercase' gibt 'Text' als Kleinbuchstaben zurück.

Num

Num (Expression) * -> Numerisch

Parameter	Typ	Beschreibung
Expression	String | Boolean	Text, der in eine Zahl umgewandelt werden soll oder Boolean, um 0 oder 1 zurückzugeben
Funktionsergebnis:	Numerisch	Zahlenwert des Textes oder Boolean Wert

Beschreibung
'Num' gibt den Zahlenwert zurück, der bei der Umwandlung des in 'Expression' übergebenen String oder Boolean entsteht.
Ist der Ausdruck 'true' wird 1 zurückgegeben, bei 'false' 0.

Position

Position (Find; String) -> Numerisch

Parameter	Typ	Beschreibung
Find	String	Zu suchender Text
String	String	Text, in dem gesucht werden soll
Funktionsergebnis:	Numerisch	Position des ersten Auftretens

Beschreibung
'Position' sucht die Position des ersten Auftretens von 'Find' in 'String' und gibt diesen Wert zurück. Wurde Suchtext nicht gefunden, liefert 'Position' 0 zurück.

Replace string

Replace string (Source; Old; New{; Size}) -> String)

Parameter	Typ	Beschreibung
Source	String	Text, in dem Teile ersetzt werden sollen
Old	String	Zu ersetzende Zeichen
New	String	Zu ersetzender Text
Size	Numerisch	Anzahl der Ersetzungen von 'Old'. Ohne Angabe werden alle Vorkommen ersetzt
Funktionsergebnis:	String	Geänderter Text

Beschreibung
'Replace string' ersetzt im Text 'Source' den Text 'Old' durch 'New' und gibt den geänderten Text zurück. 'Source' bleibt unverändert. Die Anzahl der zu ersetzenden Texte wird im optionalen Parameter 'Size' festgelegt. Wird 'Size' nicht angegeben, werden alle Vorkommen ersetzt.

String

String (Expression{; Format}) * -> String)

Parameter	Typ	Beschreibung
Expression	Numerisch|Datum|Zeit	Ausdruck, der in Text umgewandelt werden soll
Format	String | Nummer	Anzeigeformat
Funktionsergebnis:	String	Zeichenkette des Ausdrucks

Beschreibung
'String' wandelt den in 'Expression' übergebenen Ausdruck in einen Text um. 'Expression' kann numerisch oder vom Typ Datum oder Zeit sein.

Substring

Substring (Source; FirstChar{; Length}) -> String

Parameter	Typ	Beschreibung
Source	String	Anfangstext
FirstChar	Numerisch	Anfangszeichen
Length	Numerisch	Anzahl Zeichen
Funktionsergebnis:	String	Teilstring von 'Source'

Beschreibung
'Substring' gibt einen String aus 'Source' ab dem Zeichen 'FirstChar' mit der Länge 'Length' zurück. Ist 'FirstChar' grösser als die Länge von 'Source', wird 'Substring' nicht ausgeführt. Ist 'FirstChar' kleiner oder gleich 0, werden von 'Source' 'Length' Zeichen zurückgeben. Wird 'Length' nicht übergeben, werden alle Zeichen ab 'FirstChar' zurückgeben.

Uppercase

Uppercase (Text) -> String

Parameter	Typ	Beschreibung
Text	Text	In Grossbuchstaben umzuwandelnder Text
Funktionsergebnis:	String	'Text' in Grossbuchstaben

Beschreibung
'Uppercase' gibt 'Text' in Grossbuchstaben zurück.

2.8 Array

Auf ein Element eines Arrays kann mit den {}-Klammern zugegriffen werden. Der Anfang eines Arrays wird mit 1 indiziert.

Beispiel
```
ARRAY INTEGER (intArray; 10) `erzeugt ein Integer-Array mit der Länge 10
i := 1
Repeat
   Alert (String (intArray{i}))
   i := i + 1
Until (i = 11)
```

ARRAY BOOLEAN

ARRAY BOOLEAN (ArrayName; Size{; Size2})

Parameter	Typ	Beschreibung
ArrayName	Array	Name des Arrays
Size	Integer	Anzahl der Elemente im Array
Size2	Integer	Zweidimensionales Array

Beschreibung
'ARRAY BOOLEAN' erzeugt im Speicher ein Array mit 'Size' Elementen vom Typ Boolean oder passt ein bestehendes Array an. 'Size2' ist optional. Ist 'Size2' angegeben, wird ein zweidimensionales Array erstellt.
Wird 'ARRAY BOOLEAN' auf ein bestehendes Array angewendet, erweitert oder verringert sich dessen Grösse. Bei einer Erweiterung werden die neuen Elemente mit 'False' initialisiert. Bei einer Verringerung gehen die gelöschten Elemente verloren.

ARRAY DATE

ARRAY DATE (ArrayName; Size{; Size2})

Parameter	Typ	Beschreibung
ArrayName	Array	Name des Arrays
Size	Integer	Anzahl der Elemente im Array
Size2	Integer	Zweidimensionales Array

Beschreibung
'ARRAY DATE' erzeugt im Speicher ein Array mit 'Size' Elementen vom Typ Date oder passt ein bestehendes Array an. 'Size2' ist optional. Ist 'Size2' angegeben, wird ein zweidimensionales Array erstellt.
Wird 'ARRAY DATE' auf ein bestehendes Array angewendet, erweitert oder verringert sich dessen Grösse. Bei einer Erweiterung werden die neuen Elemente mit dem Datum Null initialisiert. Bei einer Verringerung gehen die gelöschten Elemente verloren.

ARRAY INTEGER

ARRAY INTEGER (ArrayName; Size{; Size2})

Parameter	Typ	Beschreibung
ArrayName	Array	Name des Array
Size	Integer	Anzahl der Elemente im Array
Size2	Integer	Zweidimensionales Array

Beschreibung
'ARRAY INTEGER' erzeugt im Speicher ein Array mit 'Size' Elementen vom Typ Integer oder passt ein bestehendes Array an. 'Size2' ist optional. Ist 'Size2' angegeben, wird ein zweidimensionales Array erstellt.
Wird 'ARRAY INTEGER' auf ein bestehendes Array angewendet, erweitert oder verringert sich dessen Grösse. Bei einer Erweiterung werden die neuen Elemente mit 0 initialisiert. Bei einer Verringerung gehen die gelöschten Elemente verloren.

ARRAY LONGINT

ARRAY LONGINT (ArrayName; Size{; Size2})

Parameter	Typ	Beschreibung
ArrayName	Array	Name des Array
Size	Integer	Anzahl der Elemente im Array
Size2	Integer	Zweidimensionales Array

Beschreibung
'ARRAY LONGINT' erzeugt im Speicher ein Array mit 'Size' Elementen vom Typ LongInteger oder passt ein bestehendes Array an. 'Size2' ist optional. Ist 'Size2' angegeben, wird ein zweidimensionales Array erstellt.
Wird 'ARRAY LONGINT' auf ein bestehendes Array angewendet, erweitert oder verringert sich dessen Grösse. Bei einer Erweiterung werden die neuen Elemente mit 0 initialisiert. Bei einer Verringerung gehen die gelöschten Elemente verloren.

ARRAY REAL

ARRAY REAL (ArrayName; Size{; Size2})

Parameter	Typ	Beschreibung
ArrayName	Array	Name des Array
Size	Integer	Anzahl der Elemente im Array
Size2	Integer	Zweidimensionales Array

Beschreibung
'ARRAY REAL' erzeugt im Speicher ein Array mit 'Size' Elementen vom Typ Real oder passt ein bestehendes Array an. 'Size2' ist optional. Ist 'Size2' angegeben, wird ein zweidimensionales Array erstellt.
Wird 'ARRAY REAL' auf ein bestehendes Array angewendet, erweitert oder verringert sich dessen Grösse. Bei einer Erweiterung werden die neuen Elemente mit 0 initialisiert. Bei einer Verringerung gehen die gelöschten Elemente verloren.

ARRAY STRING

ARRAY STRING (Length; ArrayName; Size{; Size2})

Parameter	Typ	Beschreibung
Length	Integer	Länge der Zeichenkette (1...255)
ArrayName	Array	Name des Array
Size	Integer	Anzahl der Elemente im Array
Size2	Integer	Zweidimensionales Array

Beschreibung
'ARRAY STRING' erzeugt im Speicher ein Array mit 'Size' Elementen vom Typ String oder passt ein bestehendes Array an. 'Size2' ist optional. Ist 'Size2' angegeben, wird ein zweidimensionales Array erstellt.
'Length' legt die maximale Anzahl Zeichen fest, die ein Element des Arrays enthalten kann. Das Maximum beträgt 255 Zeichen.
Wird 'ARRAY STRING' auf ein bestehendes Array angewendet, erweitert oder verringert sich dessen Grösse. Bei einer Erweiterung werden die neuen Elemente mit "" (leere Zeichenkette) initialisiert. Bei einer Verringerung gehen die gelöschten Elemente verloren.

ARRAY TEXT

ARRAY TEXT (ArrayName; Size{; Size2})

Parameter	Typ	Beschreibung
ArrayName	Array	Name des Array
Size	Integer	Anzahl der Elemente im Array
Size2	Integer	Zweidimensionales Array

Beschreibung
'ARRAY TEXT' erzeugt im Speicher ein Array mit 'Size' Elementen vom Typ Text oder passt ein bestehendes Array an. 'Size2' ist optional. Ist 'Size2' angegeben, wird ein zweidimensionales Array erstellt.
Wird 'ARRAY TEXT' auf ein bestehendes Array angewendet, erweitert oder verringert sich dessen Grösse. Bei einer Erweiterung werden die neuen Elemente mit "" (leere Zeichenkette) initialisiert. Bei einer Verringerung gehen die gelöschten Elemente verloren.

Size of array

Size of array (Array) -> Numerisch

Parameter	Typ	Beschreibung
Array	Array	Array, dessen Grösse gesucht ist
Funktionsergebnis:	Integer	Anzahl Elemente im Array

Beschreibung
'Size of array' gibt die Anzahl der Elemente aus 'Array' zurück.

SORT ARRAY

SORT ARRAY (Array{; Array2; ...; ArrayN}{; > or <})

Parameter	Typ	Beschreibung
Array	Array	Zu sortierendes Array
> or <		> Sortieren in aufsteigender Reihenfolge oder < Sortieren in absteigender Reihenfolge

Beschreibung
'SORT ARRAY' sortiert ein oder mehrere Arrays. Der letzte, optionale Parameter gibt an, ob 'Array' (bis 'ArrayN') auf- oder absteigend sortiert werden soll. '>' sortiert aufsteigend; '<' sortiert absteigend. Die Arrays werden aufsteigend sortiert, falls die Sortierreihenfolge nicht angegeben wird.

Find in array

Find in array (Array; Value{; Start}) -> Numerisch

Parameter	Typ	Beschreibung
Array	Array	Array, in dem gesucht werden soll
Value	Ausdruck	Wert, der gesucht werden soll
Start	Integer	Element, bei dem die Suche beginnt
Funktionsergebnis:	Numerisch	Nummer des ersten gefundenen Elements im Array

Beschreibung
'Find in array' gibt den Index des ersten übereinstimmenden Elements zurück. Die Parameter 'Array' und 'Value' müssen vom selben Typ sein.
'Find in array' funktioniert mit numerischen Arrays oder mit Arrays des Typ Text, String, Datum, Boolean.

INSERT ELEMENT

INSERT ELEMENT (ArrayName; Position{; HowMany})

Parameter	Typ	Beschreibung
ArrayName	Array	Name des Array
Position	Integer	Position, ab der die Elemente hinzugefügt werden sollen
HowMany	Numerisch	Anzahl der Elemente, die hinzugefügt werden sollen bzw. 1 Element, falls nichts angegeben wird.

Beschreibung
'INSERT ELEMENT' fügt eine beliebige Anzahl von Elementen in 'ArrayName' ein. Die neuen Elemente werden vor 'Position' eingefügt und mit dem Standard-Wert von 'ArrayName' initialisiert. Alle Elemente nach Position werden um 1 oder 'HowMany' verschoben.
Ist 'Position' grösser als die Länge von 'ArrayName', werden die Elemente am Ende hinzugefügt.

'HowMany' gibt die Anzahl der einzufügenden Elemente an. Wird 'HowMany' nicht angegeben, wird nur ein Element eingefügt. Das Array wird automatisch in der Grösse angepasst.

DELETE ELEMENT

DELETE ELEMENT (ArrayName; Position{; HowMany})

Parameter	Typ	Beschreibung
ArrayName	Array	Name des Arrays
Position	Integer	Position, ab der Elemente gelöscht werden sollen
HowMany	Integer	Anzahl der zu löschenden Elemente bzw. 1, falls nichts angegeben wird.

Beschreibung
'DELETE ELEMENT' löscht Elemente von 'ArrayName' ab 'Position'.
'HowMany' ist optional. Wird der Parameter nicht angegeben, wird nur ein Element gelöscht, sonst 'HowMany' Elemente. Die Grösse von 'ArrayName' wird automatisch aktualisiert.

COPY ARRAY

COPY ARRAY (Source; Destination)

Parameter	Typ	Beschreibung
Source	Array	Zu kopierendes Array
Destination	Array	Array, in das kopiert werden soll

Beschreibung
'COPY ARRAY' erstellt oder überschreibt 'Destination' mit dem Typ, Inhalt und der Grösse von 'Source'. Mit diesem Befehl können keine einzelnen Elemente kopiert werden.
'Source' und 'Destination' müssen vom gleichen Typ sein. Es können nur eindimensionale Arrays kopiert werden.

Count in array

Count in array (Array; Value) -> LongInteger

Parameter	Typ	Beschreibung
Array	Array	Array für das Zählen
Value	Ausdruck	Zu zählender Wert
Funktionsergebnis:	LongInteger	Anzahl der gefundenen Einträge

Beschreibung
'Count in array' gibt zurück, wie oft 'Value' in 'Array' gefunden wurde. Entspricht kein Element 'Value', wird 0 zurückgegeben.

APPEND TO ARRAY

APPEND TO ARRAY (Array; Value)

Parameter	Typ	Beschreibung
Array	Array	Array zum Anfügen eines Elements
Value	Ausdruck	Anzufügender Wert

Beschreibung
'APPEND TO ARRAY' fügt am Ende von 'Array' ein neues Element hinzu. Als Wert wird 'Value' zugewiesen.

2.9 Operatoren

String Operatoren

Operation	Syntax	Ergibt	Ausdruck	Wert
Verbindung	String + String	String	“abc“ + „def“	„abcdef“
Wiederholung	String * Numerisch	String	“ab“ * 3	„ababab“

Numerische Operatoren

Operation	Syntax	Ergibt	Ausdruck	Wert
Addition	Zahl + Zahl	Zahl	2 + 3	5
Subtraktion	Zahl - Zahl	Zahl	3 - 2	1
Multiplikation	Zahl * Zahl	Zahl	3 * 2	6
Division	Zahl / Zahl	Zahl	5 / 2	2.5
ganzzahlige Division	Zahl \ Zahl	Zahl	5 \ 2	2
Modulo	Zahl % Zahl	Zahl	5 % 2	1
Exponent	Zahl ^Zahl	Zahl	2 ^ 3	8

Vergleichsoperatoren

Operation	Syntax	Ergibt	Ausdruck	Wert
Gleichheit	String = String	Boolean	"abc“ = "abc“	True
			"abc“ = “abd“	False
	Zahl = Zahl	Boolean	10 = 10	True
			10 = 11	False
Ungleichheit	String # String	Boolean	"abc“ # "abd“	True
			"abc“ # "abc“	False
	Zahl # Zahl	Boolean	10 # 11	True
			10 # 10	False
Grösser als	String > String	Boolean	"abd“ > "abc“	True
			"abc“ > "abc“	False
	Zahl > Zahl	Boolean	11 > 10	True
			10 > 10	False
Kleiner als	String < String	Boolean	"abc“ < "abd“	True
			"abc“ < "abc“	False
	Zahl < Zahl	Boolean	10 < 11	True
			10 < 10	False
Grösser als oder gleich	String >= String	Boolean	"abd“ >= "abc“	True
			"abc“ >= "abd“	False
	Zahl >= Zahl	Boolean	11 >= 10	True
			10 >= 11	False
Kleiner als oder gleich	String <= String	Boolean	"abc“ <= "abd“	True
			"abd“ <= "abc“	False
	Zahl <= Zahl	Boolean	10 <= 10	True
			11 <= 10	False

Boolean Operatoren

Operation	Syntax	Ergibt	Ausdruck	Wert
UND	Boolean & Boolean	Boolean	("A“ = "A“) & (15 # 3)	True
			("A“ = "B“) & (15 # 3)	False
			("A“ = "B“) & (15 = 3)	False
ODER	Boolean | Boolean	Boolean	("A“ = "A“) | (15 # 3)	True
			("A“ = "B“) | (15 # 3)	True
			("A“ = "B“) | (15 = 3)	False

Anhang

Befehlsliste (alphabetisch)

Unten stehende Tabelle bietet eine Zusammenstellung aller Katana4D Befehle. Als 'veraltet' markierte Befehle sind aus Gründen der Kompatibilität ebenfalls aufgelistet, werden aber nicht mehr gepflegt und dokumentiert und sollten daher nicht mehr verwendet werden.

Befehl	Stand-alone	veraltet	Ersatz (falls veralteter Befehl)
Abs	x		
Add to date			
AddToFile			
Alert	x		
AllAxesOff	x		
AllMotorsOff	x	x	AllAxesOff
APPEND TO ARRAY			
Arctan			
ARRAY BOOLEAN			
ARRAY DATE			
ARRAY INTEGER			
ARRAY LONGINT			
ARRAY REAL			
ARRAY STRING			
ARRAY TEXT			
Ascii			
AxisFreeze	x		
AxisOff	x		
Beckhoff_Watchdog_Disable	x		
Beckhoff_Watchdog_Enable	x		
Break	x		
Calibrate	x		
CB_ReadDigitalInput	x		
CB_SetDigitalOutput	x		
CB_SetLED	x		
Change string			
Char			
CloseGripper	x		
CloseGripperDelta	x		
CognexCam_AufgabenfeldLesen		x	-
CognexCam_FeederFlippen		x	-
CognexCam_FeederProfilLaden		x	-
CognexCam_JobLaden		x	-
CognexCam_KalibKoordinatenSenden		x	-
CognexCam_KoordinatenLesen 18		x	-
CognexCam_OnlineSetzen		x	-
CognexCam_PickSetzen		x	-

Befehl	Stand-alone	veraltet	Ersatz (falls veralteter Befehl)
CognexCam_SetComport		x	-
CognexCam_StatusLesen		x	-
Confirm			
COPY ARRAY			
Cos			
Count in array			
Current date			
Current time			
Date			
Day number			
Day of			
Dec			
DELETE ELEMENT			
Delete string			
ExitApplication			
ExitProgram	x		
Exp			
Find in array			
FreezeAllAxes	x		
FreezeAllMotors	x	x	FreezeAllAxes
GetErrorMessage			
GetGridPoint			
GetGridPointDeg			
GetGridPointEnc		x	GetGridPointDeg
GetIntVar			
GetPoint			
GetPointDeg			
GetPointEnc	x	x	GetPointDeg
GetRealVar			
GetStrVar			
IfThenElse		x	If Then Else
If Then Else	x		
INSERT ELEMENT			
Insert string			
Int	x		
IsCalibrated			
K1.2_ReadDigitalInput	x	x	CB_ReadDigitalInput
K1.2_SetDigitalOutput	x	x	CB_SetDigitalOutput
K1.2_SetLED	x	x	CB_SetLED
Length	x		
Log			
Login			
Logout			
Lowercase			
Message	x		
Milliseconds	x		

Befehl	Stand-alone	veraltet	Ersatz (falls veralteter Befehl)
Mod			
ModBusTCP_Connect	x		
ModBusTCP_Disconnect	x		
ModBusTCP_ReadBit	x		
ModBusTCP_ReadWord	x		
ModBusTCP_WriteBit	x		
ModBusTCP_WriteWord	x		
Month of			
MotorFreeze	x	x	AxisFreeze
MotorOff	x	x	AxisOff
MoveAlongPath	x		
MoveAxis	x		
MoveAxisAndWait	x		
MoveDelta			
MoveDeltaLin			
MoveInToolDirection			
MoveInToolDirectionLin			
MoveLinMultipleBlend		x	-
MoveMot	x	x	
MoveMotAndWait	x	x	
MoveRadial			
MoveRadialLin			
MoveToGridOffsetPoint	x		
MoveToGridOffsetPointLin	x		
MoveToGridPoint	x		
MoveToGridPointDelta	x		
MoveToGridPointLin	x		
MoveToGridPointLinDelta	x		
MoveToPoint	x		
MoveToPointArc	x		
MoveToPointDeg	x		
MoveToPointDelta			
MoveToPointEnc	x	x	MoveToPointDeg
MoveToPointLin	x		
MoveToPointLinDelta			
MoveToPos			
MoveToPosDeg			
MoveToPosEnc	x	x	MoveToPosDeg
MoveToPosLin			
MoveTrajectory	x		
NI_ReadBank			
NI_ReadInput			
NI_SetBank			
NI_SetOutput			
Num	x		

Befehl	Stand-alone	veraltet	Ersatz (falls veralteter Befehl)
OpenGripper	x		
OpenGripperDelta	x		
OpenWindow			
OR_Start		x	-
OR_Winner		x	-
PauseProgram	x		
Position			
ProgramState	x		
Random			
ReadActualPWM	x		
ReadActualVel	x		
ReadAxisPos	x		
ReadAxisState	x		
ReadCollisionPosLimit			
ReadCollisionPosLimitLin			
ReadCoordinates			
ReadEncoders		x	ReadCoordinates
ReadFile			
ReadInput		x	CB_ReadDigitalInput
ReadK			
ReadMaxAcc			
ReadMaxPWM			
ReadMaxVel			
ReadMotorPos	x	x	ReadAxisPos
ReadMotorStatus	x	x	ReadAxisState
ReadPositions			
ReadSensor	x		
ReadSensors		x	ReadSensor
ReadSensorsUntil		x	ReadSensor
ReceiveSerial			
RepeatUntil		x	Repeat Until
Repeat Until	x		
Replace string			
Request			
ResumeProgram	x		
Round	x		
RunApplication			
RunProgram	x		
RunUserMethod	x		
S7MPI_Connect			
S7MPI_DBReadByte			
S7MPI_DBReadDWord			
S7MPI_DBReadString			
S7MPI_DBReadWord			
S7MPI_DBWriteByte			

Befehl	Stand-alone	veraltet	Ersatz (falls veralteter Befehl)
S7MPI_DBWriteDWord			
S7MPI_DBWriteString			
S7MPI_DBWriteWord			
S7MPI_Disconnect			
S7MPI_ReadDigitalInputBit			
S7MPI_ReadDigitalInputByte			
S7MPI_ReadDigitalOutputBit			
S7MPI_ReadDigitalOutputByte			
S7MPI_ReadFlagBit			
S7MPI_ReadFlagByte			
S7MPI_ReadFlagDWord			
S7MPI_ReadFlagString			
S7MPI_ReadFlagWord			
S7MPI_WriteDigitalOutputByte			
S7MPI_WriteFlagBit			
S7MPI_WriteFlagByte			
S7MPI_WriteFlagDWord			
S7MPI_WriteFlagString			
S7MPI_WriteFlagWord			
SaveGridPoint			
SaveGridPointDeg			
SaveGridPointEnc		x	SaveGridPointDeg
SavePoint			
SavePointDeg			
SavePointEnc		x	SavePointDeg
SendSerial			
SetAcc	x		
SetCollisionDetection	x		
SetCollisionParameters	x		
SetCollisionPosLimit	x		
SetCollisionPosLimitLin	x		
SetCollisionVelLimit	x		
SetComport			
SetComport			
SetIntVar			
SetK	x		
SetKatanaAddress		x	-
SetOutput		x	CB_SetDigitalOutput
SetPWM	x		
SetRealVar			
SetStrVar			
SetVel	x		
Sin			
Size of array			
SORT ARRAY			

Befehl	Stand-alone	veraltet	Ersatz (falls veralteter Befehl)
Square root			
String	x		
Substring			
Tan			
TCP_IP_CloseConnection	x		
TCP_IP_Connect	x		
TCP_IP_OpenServer	x		
TCP_IP_Ping			
TCP_IP_Receive	x		
TCP_IP_Send	x		
TCP_IP_State			
Time			
Time string			
Trunc			
Uppercase			
Wait	x		
WaitForAxis	x		
WaitForMot	x	x	WaitForAxis
WriteFile			
Year of			